68' MICRO JOURNAL

Australia	A $4.75	New Zealand	NZ $ 6.50	**$2.95USA**
Singapore	S $9.45	Hong Kong	H $23.50	
Malaysia	M $9.45	Sweden	30:-SEK	

Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems

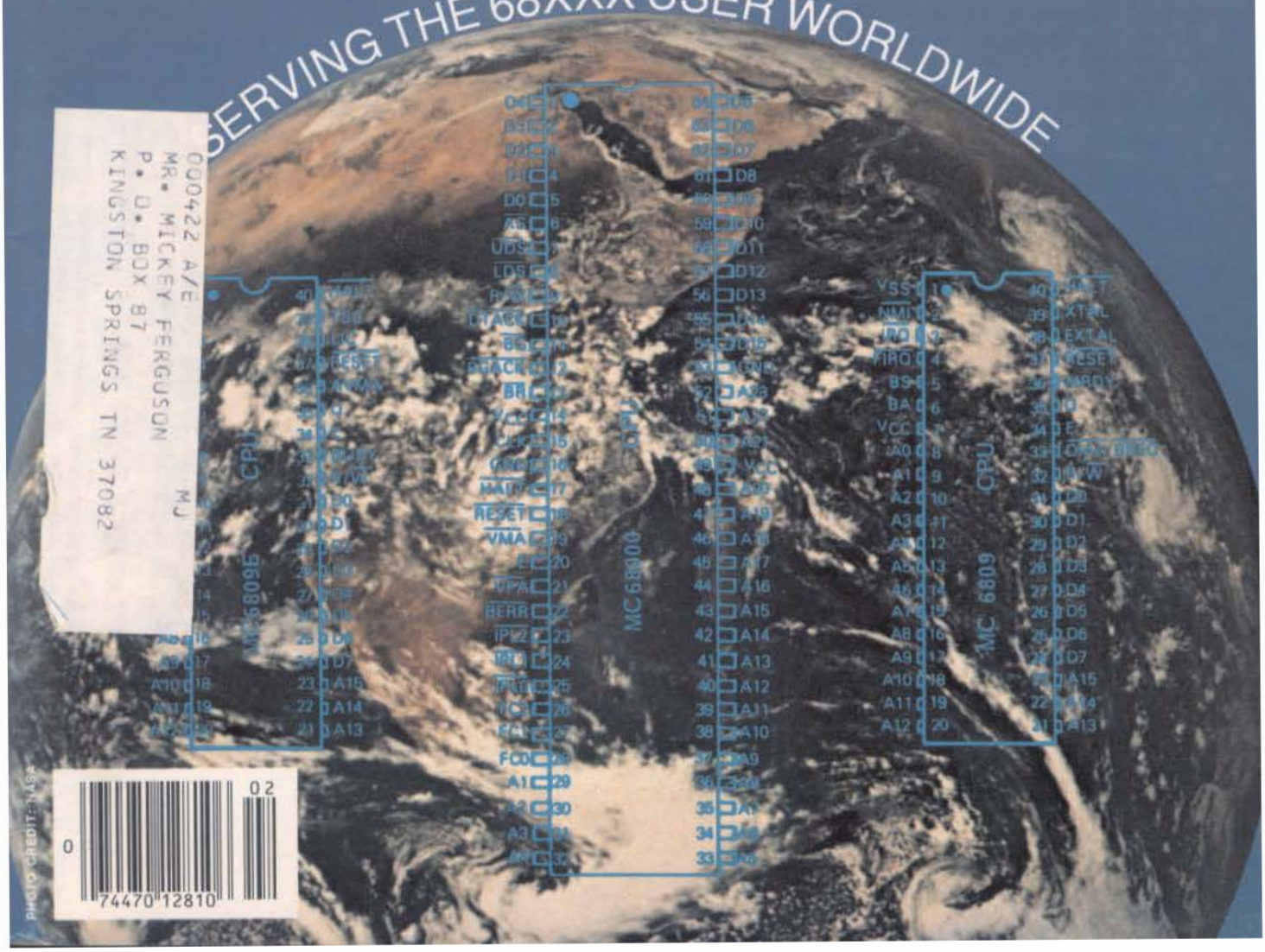
6809 68008 68000 68010 68020 68030

OS-9 The Magazine for Motorola CPU Devices FLEX SK*DOS

A User Contributor Journal

VOLUME X ISSUE II ● Devoted to the 68XXX User ● February 1988

The Grandfather of "DeskTop Publishing™"

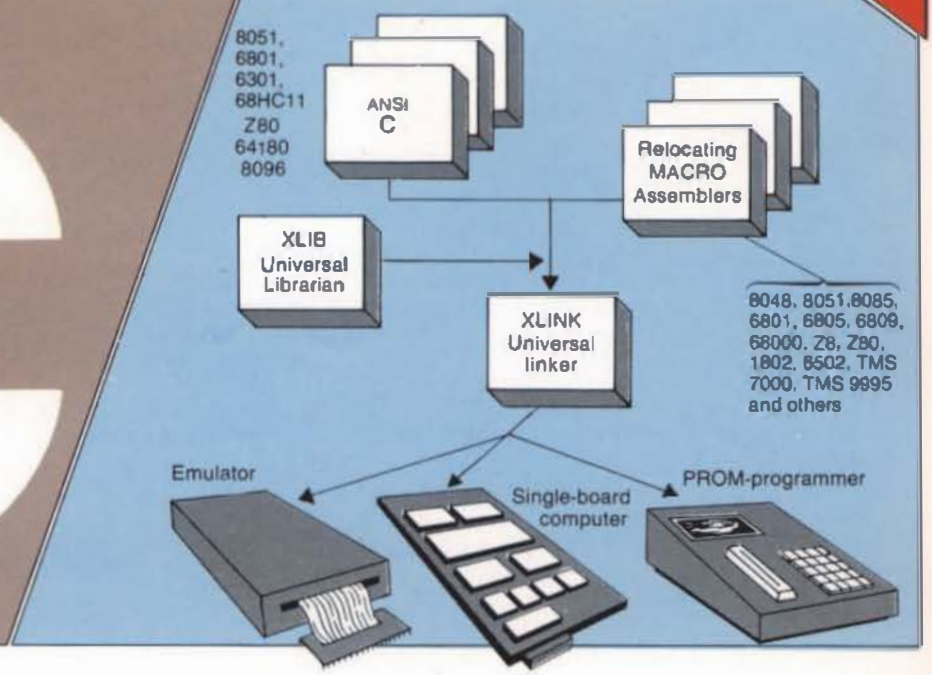


ICC C Cross Compiler
FOR OS-9/68xxx BASED SYSTEMS

NEW!

C

8051
6301/6801
68HC11
Z80/64180
8096



ANSI C

Full implementation of the proposed ANSI standard for C compilers. Includes the Kernighan & Ritchie standard plus improvements for micro-controller development.

Memory-based compiler

ICC is a fast one-pass compiler based on main memory storage. This has three major advantages:

- NO temporary files
- NO time-consuming assembly pass
- NO separate pre-processor stage

This combines into one single word: *SPEED*

PROM-able C

ICC makes it possible to put C programs into PROM, still using the full C language and all data types, including type definitions, long integers and statically initialized variables.

Built-in Type-Checking

ICC has a UNIX LINT-like type-checking option built-in into the compiler. This means that Pascal-like warnings are generated, e.g. when functions are mismatched or undeclared.

Various Options

- 8051 — single-chip
  - — 64 K CODE + DATA
  - — 64 K CODE + 64 K DATA
- 6301 and 6801
- Z80 and 64180

Full Package Development System

The ICC compiler package includes:

- C run-time library
- μ-Series Relocatable Macro Assembler
- XLINK Universal Linker
- XLIB Universal Librarian
- Floating-point support
- 150 page manual in three-ring binder

All this together give the micro-controller programmer a powerful Development System Software Environment.

For more information contact your local distributor:

Frank Hogg Laboratory
The Regency Tower
Suite 215
770 James Street
Syracuse, NY 13203
Phone (315)474-7856
Telex: 646740

Elsoft AG
Zelweg 12
CH-5405 Baden-Dattwil
Switzerland
Phone (056) 83 33 77
Telex 828275

IAR SYSTEMS

OS-9/68xxx version distribution by:

Micromaster Scandinavia AB
Box 1309, S-751 43 UPPSALA, SWEDEN.
Tel int : +46 18 13 85 95 Telex: 76129

THE GMX 020BUG DEBUGGER/DIAGNOSTIC PACKAGE

This extensive firmware package provides a broad range of program development tools and a complete suite of diagnostic programs for exercising GMX Micro-20 hardware.

The debugger includes commands for displaying and modifying registers and memory. If the optional 68881 Floating-Point Coprocessor is installed, its registers are also accessible. Memory can be displayed in hexadecimal and ASCII format, as floating-point values (single, double, extended or packed format), or as disassembled instructions (including FPC instructions). Memory modify can be done with hexadecimal values, with ASCII strings, with floating-point values, or with a one-line assembler which supports the full 68020 instruction set (although not the FPC instructions). Block move, fill, and search are also available.

Several different modes for tracing or executing user programs are provided, along with a powerful breakpoint facility. Programs and data may be downloaded from a host system or uploaded back to the host, and the GMX Micro-20 console may be used as a host system terminal. A serial printer may be hooked up, and used to make hardcopy listings of debugger sessions as desired.

The diagnostic firmware includes 90 test commands and 16 utilities. Complete test suites are provided for each functional block of the GMX Micro-20's hardware, including, for example, 9 different tests for memory, 9 tests for serial I/O ports, 2 tests for the 68881 FPC, and 9 tests for the optional memory management unit. For the peripheral control interfaces (floppy disk, SASI/SCSI hard disk or tape), test commands support a broad range of peripheral operations (read, write, format, etc.) so that the user may test both the interface and an attached device. Tests are provided for add-on I/O boards, including the ARCnet interface, I/O Channel interface, and parallel and serial expansion boards.

The utility commands allow the user to execute groups of test commands conveniently, repeat commands or command groups, enable or disable detailed fault reporting, count detected errors, or execute all the non-peripheral tests as a group. A switch option allows this last function to be invoked automatically at power-on or reset. Other utilities allow the user to check the state of the various jumpers and switches on the GMX Micro-20 directly.

In addition to the Diagnostic command package, 020Bug contains a confidence test which is always run after power-on or reset. This test does a quick checkout of the processor and the basic system elements that are needed for 020Bug operation. If any defect is found, an error code is signalled by on-and-off blinks of an LED.

DEBUGGING COMMANDS

- **MD** — Memory display
- **MM** — Memory modify
- **MS** — Memory set
- **BF** — Block fill
- **BM** — Block move
- **BS** — Block search
- **RD** — Register display
- **RM** — Register modify
- **OF** — Offset registers
- **BR** — Breakpoint set
- **NOBR** — Breakpoint delete
- **G** — Go to target code
- **GD** — Go, delete breakpoints
- **GN** — Go, stop after 1 instruction
- **GT** — Go, set temp breakpoint
- **T** — Trace
- **TC** — Trace on change of flow
- **TT** — Trace to temp breakpoint
- **LO** — Download
- **DU** — Upload
- **VE** — Verify download
- **TM** — Terminal mode
- **PA** — Printer attach
- **NOPA** — Remove printer
- **PF** — Port format
- **TD** — Time display
- **TS** — Time set
- **SD** — Switch directory
- **RS** — Restart system
- **OS** — Boot operating system

UTILITY COMMANDS

- **NV** — Non-verbose mode
- **SE** — Stop on error mode
- **LE** — Loop on error mode
- **LC** — Loop continual mode
- **ST** — Selftest mode
- **STL** — Selftest with LED mode
- **DE** — Display errors
- **ZE** — Zero errors
- **DP** — Display pass count
- **ZP** — Zero pass count
- **RL** — Read loop
- **WL** — Write loop
- **DJ** — Display baud rate jumper settings
- **DS** — Display switch
- **MJ** — Display MMU board jumper settings
- **SX** — Scan I/O expansion space

TEST COMMANDS

AN — ARCnet interface tests
- A — Wakeup test
- B — DIP Switch test
- C — Interrupts test
- D — Buffer test

CA20 - On chip cache tests
- A — Basic caching
- B — Unlike function codes
- C — Disable
- D — Clear

FD — Floppy disk tests
- A — Set parameters
- B — Drive select toggle
- C — Side select toggle
- D — Restore
- E — Seek
- F — Format track
- G — Read
- H — Write
- I — Copy buffer
- J — Compare buffer
- K — Fill buffer

IC — I/O Channel tests
- A — Print test pattern
- B — Bit rotate

MH — Miscellaneous hardware tests
- A — 68881 FPC instructions
- B — 68881 FPC control functions
- C — Tick generator
- D — Interrupt sources

MT — Memory tests
- A — Set function code
- B — Set start address
- C — Set end address
- D — Random inversion test
- E — March address test
- F — Walk-a-bit test
- G — Refresh test
- H — Random byte test
- I — Program test
- J — TAS test
- K — Test 0000-1FFF
- L — Partial longword writes test

MU — Memory Management tests
- A — Map RAM data test
- B — Map RAM address test
- C — Map RAM partial write test
- D — Map RAM random data test
- E — Accessed bit reset test
- F — Address mapping test
- G — Accessed/Dirty bits test
- H — Valid/Write Enable test
- I — Task size test

PP — Parallel port tests
- A — Print test pattern
- B — Continual test bit pattern
- C — Test bit pattern for 10 sec

PX — Parallel I/O expansion board tests
- A — Data, handshake, and IRQ test
- B — P4 connector test
- C — Data and handshake toggle

SA — SASI/SCSI port with SASI device
- A — Select drive
- B — Scan data lines
- C — Restore
- D — Seek
- E — Read
- F — Write
- G — Compare buffers
- H — Fill write buffer
- I — Test interrupt
- J — Park head
- K — Format

SC — SASI/SCSI port with SCSI device
- A — Select drive
- B — Scan data lines
- C — Restore
- D — Seek
- E — Read
- F — Write
- G — Compare buffers
- H — Fill write buffer
- I — Test interrupt
- J — Stop drive
- K — Format

SI — Serial I/O tests
- A — Select DUARTs
- B — Internal loopback
- C — External loopback
- D — Baud rates
- E — Parity modes
- F — Character lengths
- G — Handshake lines
- I — BREAK detect
- J — Interrupt output
- K — Continual handshake toggle

TA — Tape drive tests
- A — Rewind
- B — Read
- C — Write
- E — Compare buffers
- F — Fill write buffer
- G — Erase

GMX Inc. 1337 W. 37th Place, Chicago, IL 60609
(312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.
"Over a Decade of Service"
"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343
Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

MUSTANG -020 Super SBC™

DATA-COMP Proudly Presents the First Under $4300 "SUPER MICRO" See other advertising (backcover) for economy system (68008) - under $2400 complete!

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $4300, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 32serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $2500. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

Available 12.5- 25 Mhz systems, call for special prices

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based debugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

Data-Comp Division

CPI A Decade of Quality Service"

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 * *	5.1	3.2
68000 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.3*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
        {
            register long i;
            for (i=0; i < 999999; ++0);
        }
```

Estimated MIPS - MUSTANG-020 — 4.5 MIPS,

Burst to 8 - 10 MIPS: Motorola Specs

Mustang Specifications

12.5 Mhz (optional) 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
Zeroed 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Mustang Hi-Speed Systems Only from Data-Comp Div. 68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

OS-9

OS-9 Professional Ver *Includes C Compiler	$850.00
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Pascal 77	750.00
Microware Pascal	500.00
Oregonsoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Scalpel+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Pascal 77	450.00
Scalpel+ (see below)	995.00
Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	75.00
Add for 20 Mhz 68020/RAM	750.00
16 Port ser. RS-232	335.00
Requires 1 or 2 Adapter Cards below.RS232 Adapter	165.00
Each card supports 4 additional ser. ports (total of 36 serial ports supported)	
60 line Parallel I/O card	398.00
Uses 3 68230 interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	
Prototype Board	75.00
room for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	
SBC-AN	475.00
Interface between the system and ARCNET certified token passing LAN, fiber optics optional - call.	
LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Scalpel+ $695.00. SAVE $300.00

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

The PRO!

Only the "PRO" Version of OS-9 Supported!

This is *HEAVY DUTY* Country!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatibility and low cost is a must.

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
Total:		$5299.80

Save $1000.00

Complete 25 Mbyte HD System $4299.80

85Mbyte HD System $5748.80

Note: Only Professional OS-9 Now Available (68020 Version)
Includes ($500) C Compiler - 68020 & 68881 Supported -
For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division
A Decade of Quality Service
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Lama Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter continues the discussion and presentation of a public-domain portable math library written in C by Fred Fish.

MATH LIBRARY

The cexp.c function returns the complex natural exponential of its argument.

```
/*
 *       cexp    complex double precision
expohential
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX cexp (z)
COMPLEX z;
{
    COMPLEX result;
    double    expx;
    extern double    exp(), sin(), cos();

    ENTER ("cexp");
      DEBUG4 ("cexpin", "arg %le %le",
z.real, z.imag);
    expx = exp (z.real);
    result.real = expx * cos (z.imag);
    result.imag = expx * sin (z.imag);
    DEBUG4 ("cexpout", "result %le %le",
result.real, result.imag);
    LEAVE ();
    return (result);
}
```

The clog.c function returns the complex natural logarithm of its argument.

```
/*
 *    clog  complex double precision natural
logarithm
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX clog (z)
COMPLEX z;
{
    double    temp;
    extern double    cabs (), atan2(), log
();

    ENTER ("clog");
      DEBUG4 ("clogin", "arg %le %le",
z.real, z.imag);
    temp = log (cabs (z));
    z.imag = atan2 (z.real, z.imag);
    z.real = temp;
    DEBUG4 ("clogout", "result %le %le",
z.real, z.imag);
    LEAVE ();
    return (z);
}
```

The cmult.c function returns the complex product of its arguments.

```
/*
 *       cmult    double precision complex
multiplication
 */
```

```
#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX cmult (z1, z2)
COMPLEX z1;
COMPLEX z2;
{
    COMPLEX result;

    ENTER ("cmult");
     DEBUG4 ("cmultin", "argl %le %le",
z1.real, z1.imag);
     DEBUG4 ("cmultin", "arg2 %le %le",
z2.real, z2.imag);
     result.real = (z1.real * z2.real) -
(z1.imag * z2.imag);
     result.imag = (z1.real * z2.imag) +
(z2.real * z1.imag);
    DEBUG4 ("cmultout", "result %le %le",
result.real, result.imag);
    LEAVE ();
    return (result);
}
```

The cos.c function returns the cosine of its argument.

```
/*
 *      cos    double precision cosine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static double cos_pcoeffs[] =
{
    0.12905394659037374438e7,
   -0.37456703915723204710e6,
    0.13432300986539084285e5,
   -0.11231450823340933092e3
};

static double cos_qcoeffs[] =
{
    0.12905394659037373590e7,
    0.23467773107245835052e5,
    0.20969518196726306286e3,
    1.0
};

static char funcname[] = "cos";

double cos (x)
double x;
{
    auto double y;
    auto double yt2;
    auto double rtnval;
    extern double mod ();
    extern double sin ();
    extern double sqrt ();
    extern double poly ();
    struct exception xcpt;

    DBUG_ENTER (funcname);
    DBUG_3 ("cosin", "arg %le", x);
    if (x < -PI || x > PI)
    {
        x = mod (x, TWOPI);
        if (x > PI)
        {
            x = x - TWOPI;
        }
        else
        if (x < -PI)
        {
            x = x + TWOPI;
        }
    }
    if (x > HALFPI)
    {
        xcpt.retval = -(cos (x - PI));
    }
    else
    if (x < -HALFPI)
    {
        xcpt.retval = -(cos (x + PI));
    }
    else
    if (x > FOURTHPI)
    {
        xcpt.retval = sin (HALFPI - x);
    }
    else
    if (x < -FOURTHPI)
    {
        xcpt.retval = sin (HALFPI + x);
    }
    else
      if (x < X6_UNDERFLOWS && x > -
X6_UNDERFLOWS)
    {
       xcpt.retval = sqrt (1.0 - (x * x));
    }
    else
    {
        y = x / FOURTHPI;
        yt2 = y * y;
       xcpt.retval = poly (3, cos_pcoeffs,
yt2) /
            poly (3, cos_qcoeffs, yt2);
    }
```

```
        DBUG_3  ("cosout",  "result  %le",
xcpt.retval);
    DBUG_RETURN (xcpt.retval);
}
```

The cosh.c function returns the hyperbolic cosine of its argument.

```
/*
 *      cosh   double precision hyperbolic
cosine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static char     funcname[] = "cosh";

double     cosh (x)
double     x;
{
    auto struct exception xcpt;
    extern double     exp ();

    DBUG_ENTER (funcname);
    DBUG_3 ("coshin", "arg %le", x);
    if (x > LOGE_MAXDOUBLE)
    {
        xcpt.type = OVERFLOW;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
           fprintf (stderr, "%s: OVERFLOW
error\n", funcname);
            errno = ERANGE;
            xcpt.retval = MAXDOUBLE;
        }
    }
    else
    if (x < LOGE_MINDOUBLE)
    {
        xcpt.type = UNDERFLOW;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
            fprintf (stderr, "%s: UNDER-
FLOW error\n", funcname);
            errno = ERANGE;
            xcpt.retval = MINDOUBLE;
        }
    }
    else
    {
        x = exp (x);
        xcpt.retval = 0.5 * (x + 1.0 / x);
    }
      DBUG_3  ("coshout",  "result  %le",
xcpt.retval);
    DBUG_RETURN (xcpt.retval);
}
```

The crcp.c function returns the complex reciprocal of its argument.

```
/*
 *       crcp    complex double precision
reciprocal
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX crcp (z)
COMPLEX z;
{
    double     denom;

    ENTER ("crcp");
      DEBUG4  ("crcpin",  "arg  %le  %le",
z.real, z.imag);
    denom = (z.real * z.real) + (z.imag *
z.imag);
    if (!denom)
    {
        pmlerr(CRCP_OF_ZERO);
        z.real = MAX_POS_DBLF;
        z.imag = 0.0;
    }
    else
    {
        z.real /= denom;
        z.imag /= -denom;
    }
    DEBUG4 ("crcpout", "result %le %le",
z.real, z.imag);
    LEAVE ();
    return (z);
}
```

The csin.c function returns the complex sine of its argument.

```
/*
 *     csin   complex double precision sine
 */
```

```
#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX csin (z)
COMPLEX z;
{
    COMPLEX result;
   extern double    sin(), cos(), sinh(),
cosh();

    ENTER ("csin");
      DEBUG4 ("csinin", "arg %le %le",
z.real, z.imag);
     result.real = sin (z.real) * cosh
(z.imag);
     result.imag = cos (z.real) * sinh
(z.imag);
    DEBUG4 ("csinout", "result %le %le",
result.real, result.imag);
    LEAVE ();
    return (result);
}
```

The csinh.c function returns the complex hyperbolic sine of its argument.

```
/*
 *      csinh   complex double precision
hyperbolic sine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX csinh (z)
COMPLEX z;
{
    COMPLEX cexpmz;
    extern COMPLEX cexp ();

    ENTER ("csinh");
      DEBUG4 ("csinhin", "arg %le %le",
z.real, z.imag);
    cexpmz.real = -z.real;
    cexpmz.imag = -z.imag;
    cexpmz = cexp (cexpmz);
    z = cexp (z);
    z.real -= cexpmz.real;
    z.imag -= cexpmz.imag;
    z.real *= 0.5;
    z.imag *= 0.5;
    DEBUG4 ("csinhout", "result %le %le",
z.real, z.imag);
    LEAVE ();
    return (z);
}
```

The csqrt.c function returns the complex square root of its argument.

```
/*
 *   csqrt  complex double precision square
root
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX csqrt (z)
COMPLEX z;
{
    double    root, q;
    extern double    dabs(), sqrt(), cabs
();

    ENTER ("csqrt");
      DEBUG4 ("csqrtin", "arg %le %le",
z.real, z.imag);
    if (z.real || z.imag)
    {
        root = sqrt (0.5 * (dabs (z.real)
+ cabs (z)));
        q = z.imag / (2.0 * root);
        if (z.real >= 0.0)
        {
            z.real = root;
            z.imag = q;
        }
        else
        if (z.imag < 0.0)
        {
            z.real = -q;
            z.imag = -root;
        }
        else
        {
            z.real = q;
            z.imag = root;
        }
    }
    DEBUG4 ("csqrtout", "result %le %le",
z.real, z.imag);
    LEAVE ();
    return (z);
}
```

The csubt.c function returns the result of subtracting its second argument from its first argument.

```
/*
 *      csubt   double precision complex
subtraction
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX csubt (z1, z2)
COMPLEX z1;
COMPLEX z2;
{
    ENTER ("csubt");
      DEBUG4 ("csubtin", "arg %le %le",
z1.real, z1.imag);
     DEBUG4 ("csubtin", "arg2 %le %le",
z2.real, z2.imag);
    z1.real -= z2.real;
    z1.imag -= z2.imag;
    DEBUG4 ("csubtout", "result %le %le",
z1.real, z1.imag);
    LEAVE ();
    return (z1);
}
```

The ctan.c function returns the complex tangent of its argument.

```
/*
 *   ctan  complex double precision tangent
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX ctan (z)
COMPLEX z;
{
    COMPLEX ccosz;
    extern COMPLEX ccos (), csin (), cdiv
();

    ENTER ("ctan");
      DEBUG4 ("ctanin", "arg %le %le",
z.real, z.imag);
    ccosz = ccos (z);
    if ((!ccosz.real) && (!ccosz.imag))
    {
        z.real = MAX_POS_DBLF;
        z.imag = 0.0;
    }
    else
    {
        z = csin (z);
        z = cdiv (z, ccosz);
    }
    DEBUG4 ("ctanout", "result %le %le",
z.real, z.imag);
    LEAVE ();
    return (z);
}
```

The ctanh.c function returns the complex hyperbolic tangent of its argument.

```
/*
 *      ctanh   complex double precision
hyperbolic tangent
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX ctanh (z)
COMPLEX z;
{
    COMPLEX result;
    extern COMPLEX cexp (), cdiv ();

    ENTER ("ctanh");
      DEBUG4 ("ctanhin", "arg %le %le",
z.real, z.imag);
    result.real = -2.0 * z.real;
    result.imag = -2.0 * z.imag;
    result = cexp (result);
    z.real = 1.0 - result.real;
    z.imag = -result.imag;
    result.real += 1.0;
    result = cdiv (z, result);
    DEBUG4 ("ctanhout", "result %le %le",
result.real, result.imag);
    LEAVE ();
    return (result);
}
```

The dabs.c function returns the absolute value of its argument.

```
/*
 *    dabs  double precision absolute value
 */
```

```
#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static char    funcname[] = "dabs";

double    dabs (x)
double    x;
{
    DBUG_ENTER (funcname);
    DBUG_3 ("dabsin", "arg %le", x);
    if (x < 0.0)
    {
        x = -x;
    }
    DBUG_3 ("dabsout", "result %le", x);
    DBUG_RETURN (x);
}
```

The dint.c function returns the integer portion of its argument.

```
/*
 *    dint  double precision integer portion
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

#define W1_FBITS  24        /* (NOTE HIDDEN
BIT NORMALIZATION)      */
#define W2_FBITS  32          /* Number of
fractional bits in word 2 */
#define WORD_MASK 0xFFFFFFFF /* Mask for
each long word of double    */

static union
{
    double    dval;
    long    lval[2];
}   share;

double    dint(x)
double    x;
{
    int    exponent, xexp(), fbitdown;
    register long    *lpntr;

    lpntr = &share.lval[0];
    share.dval = x;
    if ((exponent = xexp(x)) <= 0)
    {
        share.dval = 0.0;
    }
    else
    if (exponent <= W1_FBITS)
    {
       *lpntr++ &= (WORD_MASK << (W1_FBITS
- exponent));
        *lpntr++ = 0;
    }
    else
    if (exponent <= (fbitdown = W1_FBITS +
W2_FBITS))
    {
        lpntr++;
       *lpntr++ &= (WORD_MASK << (fbitdown
- exponent));
    }
    else
    {
        pmlerr(DINT_2BIG);
    }
    return (share.dval);
}
```

The exp.c function returns the natural exponential of its argument.

```
/*
 *      exp   double precision exponential
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

#define C 20.8137711965230361973 /* Poly-
nomial approx coeff. */
#define D 1.0                  /* Polynomial
approx coeff. */
#define E 7.2135034108448192083  /* Poly-
nomial approx coeff. */
#define F 0.057761135831801928   /* Poly-
nomial approx coeff. */

static double    fpof2tbl[] =
{
   1.00000000000000000000, /* 2 ** 0/16  */

   1.04427378242741384020, /* 2 ** 1/16  */

   1.09050773266525765930, /* 2 ** 2/16  */

   1.13878863475669165390, /* 2 ** 3/16  */

   1.18920711500272106640, /* 2 ** 4/16  */

   1.24185781207348404890, /* 2 ** 5/16  */

   1.29683955465100966610, /* 2 ** 6/16  */
```

```
    1.35425554693689272850, /* 2 ** 7/16  */
    1.41421356237309504880, /* 2 ** 8/16  */
    1.47682614593949931110, /* 2 ** 9/16  */
    1.54221082540794082350, /* 2 ** 10/16
*/
    1.61049033194925430820, /* 2 ** 11/16
*/
    1.68179283050742908600, /* 2 ** 12/16
*/
    1.75625216037329948340, /* 2 ** 13/16
*/
    1.83400808640934246360, /* 2 ** 14/16
*/
    1.91520656139714729380  /* 2 ** 15/16
*/
};

static char     funcname[] = "exp";

double     exp (x)
double     x;
{
    register int     y;
    register int     index;
    auto double w;
    auto double v;
    auto double a;
    auto double b;
    auto double t;
    auto double temp;
    auto double wpof2;
    auto double zpof2;
    auto double rtnval;
    extern double     dabs ();
    extern double     ldexp ();
    extern double     modf ();
    auto struct exception xcpt;

    DBUG_ENTER (funcname);
    DBUG_3 ("expin", "arg %le", x);
    if (x > LOGE_MAXDOUBLE)
    {
        xcpt.type = OVERFLOW;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
           fprintf (stderr, "%s: OVERFLOW
error\n", funcname);
            errno = ERANGE;
            xcpt.retval = MAXDOUBLE;
        }
    }
    else
    if (x <= LOGE_MINDOUBLE)
    {
        xcpt.type = UNDERFLOW;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
           fprintf (stderr, "%s: OVERFLOW
error\n", funcname);
            errno = ERANGE;
            xcpt.retval = 0.0;
        }
    }
    else
    {
        t = x * LOG2E;
        (void) modf (t, &temp);
        y = temp;
        v = 16.0 * modf (t, &temp);
        (void) modf (dabs (v), &temp);
        index = temp;
        zpof2 = (x < 0.0) ?
              (1.0 / fpof2tbl[index]) :
fpof2tbl[index];
        w = modf (v, &temp) / 16.0;
        a = C + (D * w * w);
        b = w * (E + (F * w * w));
        wpof2 = (a + b) / (a - b);
          xcpt.retval = ldexp ((wpof2 *
zpof2), y);
    }
       DBUG_3 ("expout", "result %le",
rtnval);
    DBUG_RETURN (xcpt.retval);
}
```

The frac.c function returns the fractional portion of its argument.

```
/*
 *      frac   double precision fractional
portion
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

double     frac(x)
double     x;
{
    double     dint();

    return (x - dint(x));
}
```

The math library is continued in the next chapter.

EXAMPLE C PROGRAM

Following is this month's example C program; it tests the functions in the math library which accept one real argument and return a real result.

```
/*
 *      d2d.c   test double to double math
functions
 */

#include <stdio.h>
#include "pml.h"

#define MAX_ABS_ERR 1.0e-6 /* catch only bad
errors */

static int     vflag; /* Flag for verbose
option */
static int     eflag; /* Simulate an error
to error printout */
static int     sflag; /* Flag to show final
statistics */

static double   max_abs_err = MAX_ABS_ERR;

extern double     dabs ();
extern double     acos ();
extern double     acosh ();
extern double     asin ();
extern double     asinh ();
extern double     atan ();
extern double     atanh ();
extern double     cos ();
extern double     cosh ();
extern double     exp ();
extern double     log ();
extern double     log10 ();
extern double     sin ();
extern double     sinh ();
extern double     sqrt ();
extern double     tan ();
extern double     tanh ();

/*
 *    Define all recognized test functions.
Each function
 *      must have an entry in this table,
where each
 *        entry contains the information
specified in the
 *      structure "test".
 *
 */

struct test
{             /* Structure of each function
to be tested */
   char   *name;      /* Name of the function
to test */
    double     (*func)(); /* Pointer to the
function's entry point */
   double   max_err;   /* Error accumulator
for this function */
};

static struct test tests[] =
{  /* Table of all recognized functions */

    "dabs", dabs, 0.0,    /* Absolute value
*/
    "acos", acos, 0.0,    /* Arc cosine (in
radians) */
    "acosh", acosh, 0.0, /* Hyperbolic arc
cosine (in radians) */
     "asin", asin, 0.0,     /* Arc sine (in
radians) */
    "asinh", asinh, 0.0, /* Hyperbolic arc
sine (in radians) */
    "atan", atan, 0.0,    /* Arc tangent (in
radians) */
    "atanh", atanh, 0.0, /* Hyperbolic arc
tangent (in radians) */
    "cos", cos, 0.0,      /* Cosine (argument
in radians) */
      "cosh", cosh, 0.0,     /* Hyperbolic
cosine (argument in radians) */
    "exp", exp, 0.0,      /* Exponential */

   "log", log, 0.0,     /* Natural logarithm
*/
    "log10", log10, 0.0, /* Log to base 10
*/
    "sin", sin, 0.0,      /* Sine (argument
in radians) */
    "sinh", sinh, 0.0,    /* Hyperbolic sine
(argument in radians) */
    "sqrt", sqrt, 0.0,    /* Garden variety
square root */
   "tan", tan, 0.0,     /* Tangent (argument
in radians) */
      "tanh", tanh, 0.0,     /* Hyperbolic
tangent (argument in radians) */
    NULL, NULL, 0.0       /* Function list
end marker */
};

/*
 *    main  entry point for d2d test utility
 */

main (argc, argv)
int     argc;
```

```
char    *argv[];
{
    VOID dotests ();

    DBUG_ENTER ("main");
    DBUG_PROCESS (argv[0]);
    options (argc, argv);
    dotests (argv);
    statistics ();
    DBUG_RETURN (0);
}

/*
 *    dotests  process each test from stdin directives
 */

VOID dotests (argv)
char    *argv[];
{
    char    buffer[256];    /* Directive buffer */
    char    function[64];   /* Specified function name */
    double  argument;       /* Specified function argument */
    double  expected;       /* Specified expected result */
   double   result;        /* Actual result */
    double  error;          /* Relative or absolute error */
   double   abs_err;       /* Absolute value of error */
    struct test *testp;     /* Pointer to function test */
     struct test *lookup (); /* Returns function test pointer */
    register char   *strp; /* Pointer to next token in string */
    extern char    *strtok ();
    extern double   atof ();

    DBUG_ENTER ("dotests");
    while (fgets (buffer, sizeof(buffer), stdin))
    {
        strcpy (function, "(null)");
        argument = 0.0;
        expected = 0.0;
        sscanf (buffer, "%s %le %le",
                function, &argument, &expected);
        testp = lookup (function);
        if (testp == NULL)
        {
            fprintf (stderr, "%s: unknown function \"%s\".\n",
                argv[0], function);
        }
        else
        {
                    result = (*testp->func)(argument);
            if (vflag)
            {
                printf ("%s(%le) = %30.23le.\n",
                function, argument, result);
            }
            if (expected)
            {
              error = (result - expected) / expected;
            }
            else
            {
                error = result;
            }
            if (error < 0.0)
            {
                abs_err = -error;
            }
            else
            {
                abs_err = error;
            }
            if ((abs_err > max_abs_err) || eflag)
            {
                fprintf (stderr,
                        "%s: error in \"%s\"\n", argv[0], function);
                fprintf (stderr,
                "\targument\t%25.20le\n", argument);
                fprintf (stderr,
                "\tresult\t\t%25.20le\n", result);
                fprintf (stderr,
                "\texpected\t%25.20le\n", expected);
            }
            if (abs_err > testp->max_err)
            {
               testp->max_err = abs_err;
            }
        }
    }
   DBUG_VOID_RETURN;
}

/*
 *    options  process command line options
 */
```

```
options (argc, argv)
int    argc;
char    *argv[];
{
    register int    flag;
    extern int    getopt ();
    extern char    *optarg;

    DBUG_ENTER ("options");
    eflag = sflag = vflag = FALSE;
     while ((flag = getopt (argc, argv,
"#:el:sv")) != EOF)
    {
        switch (flag)
        {
        case '#':
            DBUG_PUSH (optarg);
            break;
        case 'e':
            eflag = TRUE;
            break;
        case 'l':
                  sscanf (optarg, "%le",
&max_abs_err);
           DBUG_3 ("args", "max_abs_err =
%le", max_abs_err);
            break;
        case 's':
            sflag = TRUE;
            break;
        case 'v':
            vflag = TRUE;
            break;
        }
    }
    DBUG_VOID_RETURN;
}

/*
 *      loopup   lookup test in known test
list
 */

struct test *lookup (funcname)
char    *funcname;
{
    struct test *testp;
    struct test *rtnval;

    DBUG_ENTER ("lookup");
    rtnval = (struct test *) NULL;
     for (testp = tests; testp ->name &&
!rtnval; testp++)
    {
            if (!strcmp (testp ->name,
funcname))
        {
            rtnval = testp;
        }
    }
    DBUG_RETURN (rtnval);
}

/*
 *     statistics   print final statistics
if desired
 */

statistics ()
{
    struct test *tp;

    DBUG_ENTER ("statistics");
    if (sflag)
    {
        for (tp = tests; tp ->name; tp++)
        {
           printf ("%s:\tmaximum relative
error %le\n",
            tp ->name, tp ->max_err);
        }
    }
    DBUG_VOID_RETURN;
}
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL

GETTING THE DE-BUGS OUT

Although my title is a bit of a pun, there is some truth to it. Any typeof problemoccurring in a program is called BUG. The term goes back to the days when computers were housed in rooms and sometimes buildings. The hardware consisted of tubes and relays. Programming was done by "hardwiring". Wires were physically connected from location to location. Building blocks of the program were literally connected together. If any of you are familiar with the analog computer, you will understand the technique. When something went wrong, a search was made. It could be some type of miswiring or a blown tube. During one search a moth was found between two electrical contacts. The problem was termed a "bug". Hence, the routine became getting out the bugs or debugging ones program.

So much for old stories. Now a days debugging a program means to search for problems in it operation. Seldom is the cause a fried insect stuck between two contacts. Errors come in the form of misguided program execution, syntax errors and illegal operations. Some errors are easier to find then others. Others remain extremely allusive. Some will show up at compile time. Others will occur during program execution. And there is the case when everything will compile correctly and run, but it won't be doing what was intended.

There are numerous techniques to find the bugs. Some can be done without the aide of the computer. Some require running the program. Others will actively print the progress of the program or examine it while it runs.

The oldest method is the "walk through". This involves sitting down with a listing and actually stepping through the program by hand. This method goes back the days when programmers would punch up their programs on cards. The cards were taken to a facility where they would be processed. Usually overnight the program would compiled and executed. The next day the cards and data would be picked up. If the programmer were fortunate, his data was what was expected. Most times, it was not. Some error was noted. Now the trick was to step through program looking for the problem. If luck were with him, he might find it before the day was out and resubmit his cards for reprocessing. This could take time, but it taught patience and perseverance.

Now there is interactive computing. The days are behind ofhaving to submit batch jobs. However, this method does have its merits. Many times I have created a printout of the program. Then I have sat back and carefully walked through the program by hand. At each program step, I have mentally calculated what the computer would be doing. This may mean

some calculation or an I/O. It could be a branch to some other program location. If done carefully a potential bug may be found. Ideally this method is used when a known problem exists and its approximate location is known. For a long program, doing a complete walk through would be time consuming. Boredom may occur before an error is found.

Another method is to add a line in the program displaying what it is doing. This is usually a simple statement, printing the variable of interest. It could also be a statement of where the program is currently located. Here is one way to tell where the program flow is going. I use this method quite a bit in my C programming. Take the following case:

```
#define DEBUG 1

#if DEBUG
printf("a=%d b=%d c=%d\n",a,b,c);
#endif
```

Now when I compile this, the line becomes part of the program. When the program flow gets to this line, I see the value of a, b, and c. I may use many lines like this. Always using the #if DEBUG...#endif structure. When I am ready to compile the finished product, I change the first line to:

```
#define DEBUG 0
```

Now everything that appears in the "if DEBUG" structure is no longer part of the program.

This technique can be applied to other languages. In Pascal you might want to start with a declaration like:

```
CONST
   debug = TRUE;;
```

Now to print a, b and c as before we enter:

```
IF debug
   writeln('a=',a,' b=',b,' c=',c);
```

Later when finished with debugging the program, the TRUE is changed to FALSE. Then all the debug statements will not print.

The best method to debug a program is to be able to monitor its progress. This can include setting break points, stepping through the program, examining variables, printing the program's progress and levels of "call nesting". Such a feat may sound a little complex, especially in a language that compiles down to object code. But it can be implemented and is in many systems.

Some changes in the compiling methods would be necessary. Line numbers would have to be added to the code. This could be accomplished by adding a variable, LINNUM. Then each new line that is executed could be referenced. So just before line 5 would execute, the following would be done.

```
ldx #5
stx >LINNUM
```

After the line has executed and an error flag could be checked. If an error has occurred, the error could be printed. It may look something like:

```
ldy >ERRFLAG
beq CONT
bsr PRINTERR
CONT equ *
```

Here if ERRFLAG contains a 0 execution goes to CONT. Otherwise PRINTERR is called and the error type is printed.

A symbol table could be maintained. It would have variable names and their relative locations in the data area. Whenever a variable would be examined, the debugger would check the table. Then it would go to the variable position in the data area and display its contents in the appropriate form.

Provisions could be made to use break points. This could be done by putting software interrupts in the actual code to stop its execution. Since lines are numbered, they could be executed one at a time. Using the line numbers and a listing, the program lines could be executed and displayed.

The ideas for my hypothetical debugger are implemented in many systems. In fact, systems that offer debuggers have ones that are far more complex then what I have outlined. In such systems, programs are compiled in debug or non-debug mode. So once, everything is running fine, the program can be compiled in non-debug mode.

DEBUGGING IN BASIC09

There is a debugger available in BASIC09. It does many of the things that I have outlined. The BASIC09 Debugger is a symbolic debugger. It allows interaction with the program using names and statements found in the program. If you have worked with standard BASICs, you are familiar with interactive programming. Many commands can be entered directly from the keyboard. The BASIC09 Debugger is much more advanced than this, as we will see in a minute.

To get started with it you must be have some program that is of interest. There three ways to get into the debugger mode. They are:

1. Encountering an error.
2. Executing a pause.
3. Typing a CTRL-C during program execution.

I like to use PAUSE. It is the most controllable of the three.

Let's look at a simple example. The following listing is of the procedure test.

```
PROCEDURE test
DIM i,j,k : INTEGER
PAUSE
i:=2
j:=i*i
k:=j*i+j
END
```

Now if we RUN this little, program it will appear something like this:

```
BREAK: PROCEDURE test
D:
```

The "D:" prompt is waiting for some input. Nothing has happened up to this point. The variables have not been equated to anything. So we should execute the first statement.

```
D:STEP
BREAK: PROCEDURE test
```

We have now stepped up one program step. We can now try:

```
D:PRINT i
2
```

and it will print the value of i. Next we enter:

```
D:STEP 2
BREAK: PROCEDURE test
D:PRINT j,k
4      12
```

Here we stepped up 2 lines and printed the resulting value of j and k.

This can be useful. But it does have its drawbacks. We really do not know where we are at. Every time we STEP, it prints only that a BREAK in PROCEDURE test has occurred. There is no mention of where in test. In a large program it could be very confusing. So, in the original program we change the line PAUSE to read TRON. This is the TRace ON feature. Now run the procedure test.

```
*0011    i:=2
=2
*0018    j:=i*i
=4
*0024    k:=i*j+j
=12
*0034    END
```

This output is a little more useful. First, notice how it printed the program lines with their I-Code line numbers. We see exactly what has occurred. Second, the result of the line is printed. This means no longer having to use the PRINT statement.

Let's look at another simple example. This one will be test2. Here it is.

```
PROCEDURE test2
DIM i,j:INTEGER
TRON
FOR i:=1 TO 2
j:=altered(100)
IF MOD(j,2)=0 THEN
PRINT "EVEN"
ELSE
PRINT "ODD"
ENDIF
NEXT i
END
```

This a simple program. It generates random numbers and test whether they are odd or even. The result of the test is printed. Running it will give the following printout.

```
*000D    FOR i:=1 TO 2
=1
=2
*001D    j:=altered(100)
=42
*0027    IF MOD(j,2)=0 THEN
=True
*0035
*0036    PRINT "EVEN"
=EVEN
EVEN
*003E    ELSE
*004B    NEXT i
*001D    j:=altered(100)
=79
*0027    IF MOD(j,2)=0 THEN
=False
*0042    PRINT "ODD"
=ODD
ODD
*0049    ENDIF
*004B    NEXT i
*0056    END
```

Notice how the result of each line is printed. In the IF MOD(j,2)=0 THEN statement, a TRUE of FALSE is printed according to the outcome. On the PRINT lines the result is also printed. After line *0036, we see:

```
=EVEN
EVEN
```

The first "EVEN" is the result of the debugger encountering it. Hence, the "=" sign precedes it. The second one is the result of the actual line. (As a side note the debugger is printing to the standard error path. The program is printing to standard output path.)

TRON and PAUSE have their counter parts. They are TROFF and CONT which are short for TRace OFF and CONTinue. TROFF discontinues the trace mode. CONT will initiate program execution. Using them, select parts of a program can be easily debugged.

During a debug session, any variable can be alterred. This is easily done using the LET command. Take the following little program.

```
PROCEDURE test3
DIM i:INTEGER
i:=3
PAUSE
PRINT i
END
```

If we were to run this and the PAUSE occurred, we could easily enter CONT and SEE the result. It would print a 3. Or we could change it with:

```
LET i:=8
CONT
```

This time an 8 would be printed. Any dimensioned variable can be changed. This is extremely handy to test the effect of altering variables. When running only a portion of a program the variables can preset.

Another useful tool is the STATE command. It prints the nesting of the procedures called. It starts with the current procedure and prints the calling procedures in reverse order. Following are procedures that call other procedures.

```
PROCEDURE test4
RUN test5
END

PROCEDURE test5
RUN test6
END
```

```
PROCEDURE test6
RUN test7
END

PROCEDURE test7
PAUSE
END
```

If we were run test4, we would stop in the debug mode in test7. Trying the STATE command we see the following.

```
D:STATE4
PROCEDURE test7
CALLED BY test6
CALLED BY test5
CALLED BY test4
```

If procedures call other procedures and an error occurs, it may be difficult to learn from which procedure it was called. But with the STATE command, a trace back of commands will provide the information.

Another command of interest is BREAK. This allows setting a break point at some procedure. I find this one useful when wanting to periodically check something in a particular procedure. I put a PAUSE in the main procedure and then enter:

```
D:BREAK test8
D:CONT
```

The main program continues execution. Whenever, test8 is encountered, execution halts. I can step through it or use CONT to restart execution. If test8 is executed again, we will stop there.

The BASIC09 debug mode, also provides some other commands. They are:

LIST List the suspended procedure
$ Call an OS-9 Shell
DIR Display procedures workspace directory
DEG Select Degrees for working with angles
RAD Select radians for working with angles
Q Quit debug session

These are self explanatory. They are similar to their counterparts in BASIC09.

Debugging a program can be a major part of creating it. Good programming technique and care can help reduce the amount of errors. But inevitably problems will arise. Finding errors and correcting them involves good debugging technique and if available, a good debugger.

PLIST, A PROGRAM LISTER

As I mentioned earlier, for a debugger to display a "list" version, some type of listing must be created. This is usually done at compile time. My assemblers and Pascal compilers do this. Basic09, however, permits a listing to made at anytime (but before PACKing). When I do C language programs, I have no such listing. In fact, in most of the C compilers I have worked with, there is no listing. So, this month I am presenting PLIST.

PLIST is a generic program lister. It can be used on any file. It has only a few functions. It adds line numbers to the start of each line. It puts in top and bottom margins. It also numbers pages and writes a header at the top of each page. The header includes the file being listed and a page number.

The syntax for using PLIST is simple. To list a file, called MYFILE, enter:

```
OS9: list myfile
```

The output will go to the CRT screen. You can also redirect it to anywhere. Most likely it will be the printer or another file.

I had some thought about adding some frills to this program. Somewhere in the header, the time and date could be added. The header and margins could be options that are selected from the input line with a -H and -M. These and other inovations, I will leave to your imagination.

Have a good month! See you next time.

LISTING

```
0000 /* ********************************
0001
0002    Name: Program LISTer
0003    Date: 28-OCT-87
0004    By: Ronald Voigts
0005    To Compile:  ccl plist.c
0006
0007   ********************************
0008
0009    Version 1.00     Original.
0010
0011    ********************************
0012
0013    Function:
0014       PLIST prints a formatted
0015       version of a file.  It adds
0016       numbers to lines, puts in a
0017       top margin with header and a
0018       bottom margin.
0019
0020   ********************************
*/
0021
0022 #include <stdio.h>
0023
0024 main( argc, argv )
0025 int argc;
0026 char **argv;
0027 (
0028      FILE *fopen(), *fp;
0029      char line[120];
0030      register int index=0;
0031      int page=0;
0032      int count=0;
0033
0034 /* Check for the correct number of
parameters */
0035      if ( argc != 2 ) {
0036         printf("Error in Parameter
List\n");
0037           exit(0);
0038      }
0039
0040 /* Open the file */
0041    if ( (fp=fopen(argv[1],"r")) ==
NULL )
0042           exit( errno );
0043
0044 /* Input line numbers */
0045     while ( fgets(line, 120, fp )
!= NULL ) (
0046
0047 /* Print header if necessary */
0048          if ( count++==0 )
0049        count=print_header( ++page,
argv[1] );
0050
0051 /* Print the line with number */
0052          printf("%04d %s", index++,
line);
0053
0054 /* Print footer if necessary */
0055          if ( count==60 )
0056             count=print_footer( );
0057
0058      } /* End of while */
0059
0060 /* Close the file */
0061      fclose( fp );
0062
0063 } /* End of main program */
0064
0065 /* Header routine */
0066 int print_header( p, s )
0067 int p;
0068 char *s;
0069 (
0070      print_cr( 2 );
0071           printf("Program  Lister
Version 1.0");
0072    printf("                    Page
%3d\n", p );
0073      printf("File name: %s\n", s );
0074      print_cr( 3 );
0075      return( 7 );
0076 } /* End of print_header() */
0077
0078
0079 /* Footer routine */
0080 int print_footer()
0081 (
0082      print_cr( 5 );
0083      return( 0 );
0084 } /* End of print_footer */
0085
0086 /* Print End-of-Lines */
0087 print_cr( n )
0088 int n;
0089 {
0090      register int i;
0091
0092      for ( i=0; i<n ; i++ )
0093           printf("\n");
0094 } /* End of print_cr() */
```

+++

The Macintosh™ Section
Reserved as a
A place for your thoughts
And ours.....

Mac-Watch

"Solutions"... A review of SuperScrap and Clipper

By James E. Law
1806 Rock Bluff Rd.
Hixson, TN 37343

A major category of software for the MacIntosh is branded as "utilities". While such programs are not usually as sexy as desktop publishing packages, they do make the user's life simpler. This month I want to review two such offerings from Solutions, Inc. These include SmartScrap (a scrapbook upgrade) and Clipper (for sizing graphics).

SmartScrap

The MacIntosh scrapbook is a nifty idea. It provides for saving frequently-used text and graphics in a system file that is available in any application through a desk accessory. Letterhead graphics or boilerplate language for letters can be kept readily available for quick use without quitting the current application.

The original scrapbook is not without limitations. Only one such file is generally allowed and it has to be in the system folder of the controlling system. The size of the window is fixed with no zooming or scrolling allowed. Consequently, only a portion of the selected text or graphics can be viewed if it is bigger than the window. Also, cutting and copying involves all or none of the selected image. Portions of the image cannot be selected.

SmartScrap removes the above limitations and substantially improves the usefulness of the scrapbook function. Any number of SmartScrap files can be set up and placed anywhere on the startup disk or disks in internal drives. These files can be accessed through a 31K disk accessory which provides for creating new files and switching between different scrapbook files.

Images are pasted into SmartScrap just as they were into the old scrapbook but once there, they can be handled much easier. A zoom button expands the SmartScrap window to fill the screen. Horizontal and vertical scrolling is provided for graphics. (Only vertical scrolling is allowed for text.) A selection rectangle can be used to copy (but not cut) any portion of a selected graphics image. Automatic scrolling occurs if the item being selected with the rectangle extends off the screen. Similarly, a text tool can be used to copy any portion of a text selection in SmartScrap.

A pictorial table of contents is constructed for a scrapbook file upon request. Double clicking an image results in that image being immediately displayed. I experienced a number of problems with this feature, however. On occasion, the table of contents displayed images that were not in the scrapbook and incorrectly indicated the location of images. After a while, this problem cleared up and I could not get it to reoccur. I do not know if it was a SmartScrap bug or incompatibility with my desk accessories. I suspect the latter.

SmartScrap out classes the original scrapbook by a mile. If you use the scrapbook, scrap the original version and buy SmartScrap.

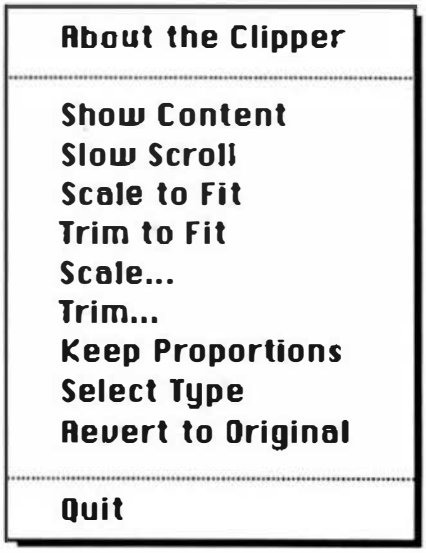
Consider the following, however, before you spend your $50 on SmartScrap or other scrapbook upgrade. Clip art management programs, like Art Roundup, can do almost everything SmartScrap does and more. Images are left in their original paint or draw type files and accessed through a desk accessory. Similar desk accessories are marketed to access and manipulate text files. In the final analysis, such programs may make the scrapbook largely obselete for many users

Clipper

Having graphics neatly placed in SmartScrap files ensures their ready availability. It does not, however, ensure that the art work is the right size for your application. Suppose that when preparing a Page Maker publication you decide to add an illustration that is the width of a column and 3 inches long. What do you do? Well, you could paste the graphic into Super Paint or MacPaint and re size it. The problem is knowing when it is the right size. If the final size is really critical, it might take several trips back and forth between Page Maker and the graphics program until you get it right. A better solution would be to resize the object in Page Maker by stretching or shrinking the graphics block with the graphics placed in its target position. This, while better than the first approach, is still a matter of trial and error.

Clipper is a desk accessory which provides an elegant solution to this problem. It provides for very accurate scaling and/or trimming of graphics. When Clipper is chosen from the DA menu, a new Clipper menu bar appears and a transparent frame is superimposed over the active application. This frame is moved into position and re sized as needed to exactly match the space to be occupied by the graphics. (This is as easy as moving and re sizing a rectangular frame in the draw portion of SuperPaint.) Tabs on the frame display the size of the area enclosed in number of pixels.

Selecting "Show Contents" from the Clipper menu displays the contents of the clipboard inside this frame. If the image is larger than the frame, only the portion inside the frame will be visible. Scroll bars will appear on the frame so that all portions of larger images can be viewed. Unless the size of the image is perfect as is, scaling or trimming is then needed.



The Clipper Menu

Clipper offers four alternatives for re sizing graphics each of which is accomplished by making proper selections from the Clipper menu.

1. The graphics can be Scaled to Fit the area enclosed by the frame. When the option is chosen, the entire graphics image is shrunk or expanded to fill the enclosed area with original proportions maintained.

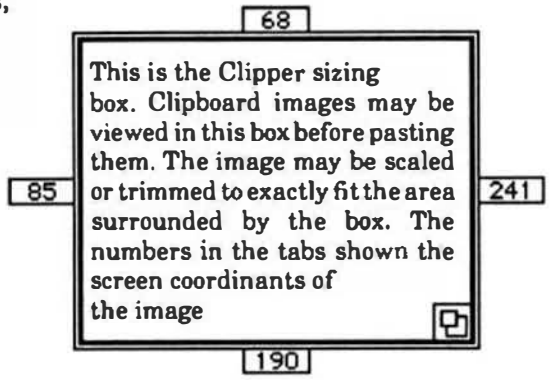
2. The graphics can be Trimmed to Fit the enclosed area. When this option is chosen, the portions of the image not visible in the Clipper frame are trimmed away.

3. If you need more control over the scaling of images, (e.g., to maximize the clarity of re sized paint-type graphics), the Scale option can be selected. This option allows separate horizontal and vertical resizing independent of the size of the Clipper frame.The scaling information may be expressed in pixels, millimeters, picas, inches, or percent.

4. Similar flexibility is available for trimming by selecting the Trim option from theClipper menu. This option allows the specific screen coordinates to be given for each side of the trimmed image.

Additional features on the Clipper menu include the ability to slow the scrolling speed for increased precision, to revert the image to its original size, and to select the type of images to be displayed and re sized. This last option can be used when a graphics contains multiple types of images (e.g. TEXT and PICT) and when only a portion of these are to be pasted into the new application. By this option, the text portion of an image could be deleted from a chart to be pasted into a report.

The above explanation probably makes Clipper sound more complicated than it really is. In reality, this DA is intuitive and, although the manual is entirely adequate, it is not needed. To illustrate how easy Clipper is to use, let's go back to the problem that started off this review, that is,



The Clipper Box

the need for a 3-inch column illustration for a Page Maker publication. Here's how Clipper would be used:

1.Select Clipper from the DA menu

2.Move and re size the Clipper frame to enclose the area to be filled by the illustration.

3.Select Show Content from the Clipper menu to display the image from the Clipboard in its original size inside the frame.

4.Select Scale to Fit or Trim to Fit from the Clipper menu and the image is instantly re sized to fill the frame.

5.Quit Clipper and paste the image. It will appear in exactly the correct size and ready to make final positioning adjustments.

This program does what it is advertised to do and does it well. In trimming and scaling a number of images for use with PageMaker and ReadySetGo, no problems were encountered. Anyone needing to precisely size graphics for placement in page layouts will appreciate the simplicity and usefulness of this program.

Final Comments

ScrapScrap and Clipper will work with any Mac with 512K or more. In both cases, the manuals are clear and complete. These products are recommended.

Other "Solutions"

I had intended to also cover another somewhat related Solutions, Inc. product called Glue. This program allows users to view ReadySetGo or PageMaker pages, Excel spread sheets, or full-sized MacPaint documents without the documents used to generate them. An upgraded product called SuperGlue is now being sold which allows the images to be converted to regular scrapbook images where they can be viewed and edited without requiring SuperGlue. I will forego a review of Glue now in the interest of providing more current information on SuperGlue later.

Logically Speaking

The following is the beginning of a continuing series. Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

The Mathematical Design of Digital Control Circuits

By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

Mile 6, heading for Mile 7.

By now you should all be thoroughly refreshed after a nice, long test-free rest, but before we take off I'd like to warn you to stay close together. Try not to straggle, as we'll be skirting the territory of the M'bul-yan tribe, and they're known to have a habit of picking off lone travellers and strays from a party. Folks who get "picked up" by the M'bul-yans never get heard from again, and to date no-one has ever been able to find out what happens to them - mainly because whoever hangs around trying to find out also never gets seen again. So be warned! Stay close!

PREPARING FOR RED-NUMBER DECODING

	0	1	2	3	4	5	6	7
Y1		1		1			1	1
Y2		1		1	1	1		
L1		1		1	Φ	1		Φ

Diagram 20

If our pencils are all sharpened and at the ready, here we go with a preliminary technique for decoding the red numbers in our flow-table in order to see what our circuit looks like. In this simple example, we'll begin by drawing up a table as shown in Diagram 20, with all the red numbers arranged in numerical order in the heading, and with rows labelled for our relays and output. The table is completed in this manner :

Let's start with the two relays, by examining our STATE-DIAGRAM again. The first box, Box-1, gives the relay-states in binary as 00, so we do nothing with this Box. Box-2 contains the code 01, the '1' indicating that we are interested in relay Y2. (Remember the bits are in the order Y1, Y2). We're at Box-2, so we therefore look in the FLOW-TABLE for all addresses containing a black-2 in Box-A. There are two such addresses, one of which has a red-4 in the corresponding Box-B, and the other a red-5. Are you with me so far? If not, go back and take this paragraph again - a little more slowly. As this red-4 and red-5 pertain to Y2, we indicate this fact by inserting a '1' in columns 4 and 5, row Y2, of Diagram 20.

Then we move on to the next Box in our STATE-DIAGRAM, Box-3-black, which contains the binary-code 11, informing us that we are interested here in both Y1 and Y2. Box-3 means we now have to look for black-3s in the FLOW-TABLE, and again we find two of them, with corresponding red numbers 1 and 3. So, in columns 1 and 3 of Diagram 20, rows Y1 and Y2, we insert '1's to record this fact.

Finally, Box-4 in the STATE-DIAGRAM, coded 10, signals that we are interested only in Y1 and black-4s. Once more we find two addresses with a black-4 in Box-A in the FLOW-TABLE, corresponding to red-6 and red-7, so in columns 6 and 7 of diagram 20, row Y1, we insert 1s to record this info.

This leaves only row L1 in Diagram 20 to be filled in. This is accomplished by systematically scanning through all Box-Cs in the Flow-Table, row by row, and recording 1s or phi's according to the red numbers in the corresponding Box-B.

You may be interested to know that the red numbers are actually minterms, and correspond to a decimal translation of the co-ordinates of the squares on a K-map which our circuit would occupy - if we went to the trouble of re-translating our networks into Boolean expressions and then plotting them on a K-map. In fact, from now on, I will often refer to a red number as a minterm - - minterm-6 or minterm-73, for instance.

NOW FOR THE ACTUAL DECODING

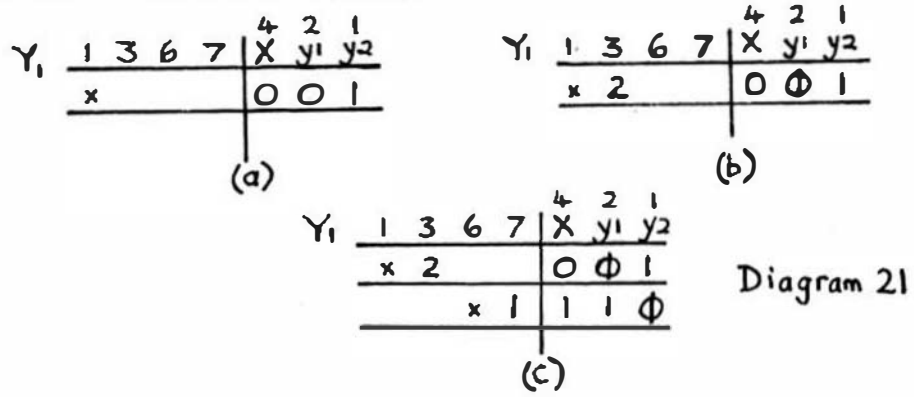


Diagram 21

Let's tackle Y1 first, shall we? For starters, we draw up a decoding-table for Y1, as shown in 21a. To the left of the vertical line our table is headed with the minterm-numbers appropriate to Y1 (see Diagram 20), and to the right with control-designations X, y1 and y2, which, in turn, are headed with their corresponding binary values 4, 2 and 1. Decoding is begun by marking a small 'x' under the lowest available minterm (in this case it's '1'), and the corresponding binary equivalent is written to the right, ie, 001.

Keeping in mind that our lowest minterm is 1, we switch our attention to the 001 to the right. Commencing with the highest bit-position (that is, 4), we see that it contains '0'. It's our intention to make this into a phi, if at all possible. This means that we'd have to superimpose a '1' over the '0' in order to create this symbol. Inserting a '1' in this bit-position corresponds to ADDING 4 to our base-minterm. 1 + 4 = 5, but unfortunately Y1 does not have a minterm-5, and we are therefore NOT allowed to change this bit-position to a phi. Stay with me, and it'll all gradually become clear!

Bit-4 being ruled out, we move right to bit-2, which also contains a 0. To convert this to a phi, we would again have to try superimposing a 1, that is, ADDING the value 2 to our base-minterm. 1 + 2 = 3, and we observe that Y1 does in fact have such a minterm in its heading. Accordingly we place a 2 under minterm-3 (to indicate that it was bit-2 that allowed us to do this), and go ahead and superimpose a 1 (thus creating a phi) in the binary number to the right. See Diagram 21b for the decoding to this point.

Finally, we consider the least-significant bit of our binary number. This contains a 1, which means that in order to convert it to a phi we'd have to superimpose a 0, not a 1 as in our earlier tries. This means having to SUBTRACT the value of this bit-position from our base-minterm, so we ask ourselves "Is 1 - 1 = 0 available?" The answer is NO, so this completes the operation on Line 1.

Because we haven't yet "covered" all of Y1's minterms, we open up a new line of decoding (see 21c for this) by placing a small 'x' under the NEXT LOWEST AVAILABLE minterm not already covered. This happens to be minterm-6, and again we write its binary equivalent to the right, that is, 110.

Then we repeat our earlier procedure, thus :

"Is 6 - 4 = 2 available? No!" Note that we SUBTRACT 4 because we're considering placing a 0 over the 1 in bit-position 4, which corresponds to REMOVING the value 4 from the binary number. Next "Is 6 - 2 = 4 available? No!" Last bit coming up, "Is 6 + 1 = 7 available? Yes!" So we enter 1 under minterm-7 to indicate that it was the 1-bit which allowed us to cover it, and change the final bit of 110 into a phi. At the risk of becoming boring, note that we ADD a 1 in this case, because we're considering superimposing a 1 on to a 0. Don't forget to convert the appropriate bit-position to phi if you successfully expand your base-minterm, otherwise your resultant circuit won't work properly!!!

This completes the decoding for Y1, as we've now covered all available minterms. Our decoding, to the the right, consists of the binary terms 0-1 and 11- (replacing phis with '-'), which translate into the Boolean expression

Y1 = Xy2 + Xy1

from which you should experience no difficulty in drawing up the corresponding network for Y1.

And so on for Y2 and L1 (shown in Diagram 22) except that in the case of L1, minterms 4 and 7 have a phi marked over them, to indicate that these minterms are OPTIONAL, and we don't have to cover them unless they should prove useful to us.

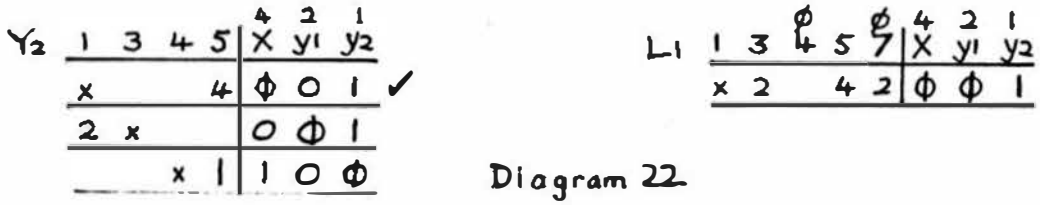


Diagram 22

The procedure for Y2 goes like this :

Place a small 'x' under the lowest available minterm, minterm-1, and write the equivalent binary value, 001, to the right. OK, here we go! "Is 1 + 4 = 5 available? Yes!" So we place a 4 under minterm-5 (because bit-4 produced this), and change bit-4 to a phi. Next "Is 1 + 2 = 3 available? Yes!" BUT .... and here is the tricky part! You'll recall from our K-map days that when we formed loops on the map (which, in essence, is what we're doing here), the size of the loops had to be related to the binary system, and we were therefore only allowed to form loops of size, 1, 2, 4, 8 ... minterms. So also with our decoding-table!

To date, in our decoding-table, we commenced with a single minterm, and then expanded it to two. Now we have three, which we MUST expand to four, or else forget this "try" altogether. How do we do this? The rule is that if a bit-number can be added to the base-minterm, THEN IT MUST ALSO BE ADDED TO EVERY OTHER MINTERM ALREADY COVERED BY THE CURRENT DECODING-LINE. Therefore, because 1 + 2 = 3 is available, but 5 + 2 = 7 is NOT, we are forbidden to make use of bit-2 at all, otherwise, as you can see, we'd be forming a loop of only 3 minterms, composed of minterms 1, 3 and 5. Further, because we "started a run" but were unable to complete it successfully, it COULD happen that this row will not be necessary to our final decoding. That is, it may NOT be an ESSENTIAL prime implicant. (See an earlier lesson for a definition). We indicate an uncompleted run by placing a 'tick' to the right of our decoding-line.

We still carry on with this line, however, and ask next "Is 1 - 1 = 0 available? No!" So that completes the decoding of this line, and we start a new row by placing a small 'x' under the next lowest available minterm not covered at all so far, namely 3, and the corresponding binary-value 011 to the right.

Proceeding, "Is 3 + 4 = 7 available? No!" "Is 3 - 2 = 1 available? Yes!" So we place a 2 (why?) under minterm-1 and change bit-2 to the right into a phi. DON'T FORGET THIS PHI OPERATION! Finally, "Is 3 - 1 = 2 available? No!" And that's the end of row 2.

However, minterm-4 is still uncovered, so we enter a small 'x' below it in row 3 and write the equivalent 100 to the right. "Is 4 - 4 = 0 available? No!" "Is 4 + 2 = 6 available? No!" "Is 4 + 1 = 5 available? Yes, it is!" And we therefore enter a 1 under minterm-5 and change bit-1 to the right into a phi.

Just as with a K-map, we can use minterms as many times as we wish during the decoding, even though they may already be covered in an earlier row. The base-minterm for starting a new row MUST, however, NOT be covered by an earlier row's decoding.

We observe that our last two (unticked, and therefore ESSENTIAL) rows actually cover all available minterms, thus row-1 is not essential to our decoding, although this will not ALWAYS be the case. Accordingly we have for Y2 the decoding :

Y2 = Xy2 + Xy1'

See Review in January 1987 issue of 68 Micro Journal

SOFTOOLS

The following programs are included in object form for *immediate* application. PL/9 source code available for customization.

READ-ME Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.
CONFIG one time system configuration.
CHANGE changes words, characters, etc. globally to any text type file.
CLEANTXT converts text files to standard FLEX, SK*DOS files.
COMMON compare two text files and reports differences.
COMPARE another check file that reports mis-matched lines.
CONCAT similar to FLEX, SK*DOS *append* but can also list files to screen.
DOCUMENT for PL/9 source files. Very useful in examining parameter passing aspects of procedures.
ECHO echos to either screen or file.
FIND an improve find command with "pattern" matching and wildcards. Very useful.
HEX dumps files in both hex and ASCII.
INCLUDE a file copy program that will accept "includes" of other disk files.
KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.
LISTDIR a directory listing program. Not super, but better than CAT.
MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted. Very fast. Very useful.
MULTICOL width of page, number of columns may be specified. A MUST!
PAGE similar to LIST but allows for a page header, page width and depth. Adjust for CRT screen or printer as set up by CONFIG. A very smart print driver. Allows printer control commands.
REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!
SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.
SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on nth word and sort on characters if file is small enough, sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.
TPROC a small but nice text formatter. This is a complete formatter and has functions not found in other formatters.
TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.
UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.
WC a word count utility. Can count words, characters or lines.

NOTE: this set of utilities consists of 6 5-1/4" disks or 2 8" disks, w/ source (PL9). 3 5-1/4" disks or 1 8" disk w/o source.
Complete set SPECIAL INTRO PRICE:
5-1/4" w/source FLEX - SK*DOS - $129.95
w/o source - $79.95
8" w/source - $79.95 - w/o source $49.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F, S and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!
Levels I & II only - OS-9 $69.95

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
Level I OS-9 obj. $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK*DOS Format so it can be used normally by FLEX, SK*DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX, SK*DOS Directory, Delete FLEX, SK*DOS Files, Copy both directions, etc. FLEX, SK*DOS users use the special disk just like any other FLEX, SK*DOS disk
O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full

set of comments and errors messages.
OS-9 $85.00

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX, SK*DOS.* It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. **Using HIER a regular (any) FLEX, SK*DOS disk (8 - 5 - hard disk) can have sub directories.** By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day solution **that all current large systems use. Each directory looks to FLEX, SK*DOS like a regular file, except they have the extension** '.DIR'. A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of FLEX, SK*DOS. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!
*FLEX - SK*DOS $79.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX, SK*DOS utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
*ALL 4 Programs (FLEX, SK*DOS, 8" or 5") $99.50*

COPYCAT from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SK*DOS, SSB DOS68, and Digital Research CP/M Disks while operating under SK*DOS , FLEX1.O, FLEX 2.O, or FLEX 9.O with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.
F, S and CCF 5" - $50.00 F, S 8" - $65.00

VIRTUAL TERMINAL from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight task on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between task at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.
O & CCO - obj. only - $49.95

FLEX, SK*DOS DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX, SK*DOS Utilities for every FLEX, SK*DOS Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.
ALL Utilities include Source (either BASIC or A.L. Source Code).
F, S and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
*FLEX, SK*DOS, CCF, OS-9, UniFLEX, 68000 & 6809*
Source $100.00 - without Source $50.00

X-TALK from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.
X-TALK Complete (cable, 2 disks) $99.95

NEW PRICES 6809 CCF and CCO - $99.95,
F, S or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES 6809 CCF and CCO - $69.95,
F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES 6809 CCF and CCO - $59.95,
F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!
F, S or O - $329.95, U - $549.95
O, 68000 $695.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DYNACALC -- Electronic Spread Sheet for the 6809 and 68000.
F, S, OS-9 and SPECIAL CCF - $200.00, U - $395.00
OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find back order or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.
F, S - $59.95, U - $89.95

CROSS ASSEMBLERS

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
*68000 or 6809, FLEX, SK*DOS, CCF, OS-9, UniFLEX*
any object or source each - $50.00
any 3 object or source each - $100.00
Set of ALL object $200.00 - w/source $500.00

XASM Cross Assemblers for FLEX, SK*DOS from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.
*Complete set, FLEX, SK*DOS only - $150.00*

CRASMB from LLOYD I/O -- Supports Motorola's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8080/85, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Length to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX, SK*DOS (binary). Written in Assembler ... e.g. Very Fast.

CPU TYPE - Price each:

For:	MOTOROLA	INTEL	OTHER	COMPLETE SET
FLEX9	$150	$150	$150	$399
SK*DOS	$150	$150	$150	$399
OS9/6809	$150	$150	$150	$399
OS9/68K	------	------	------	$432

CRASMB 16.32 from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code Format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 computer.
*FLEX, SK*DOS, CCF, OS-9/6809 $249.00*

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX, SK*DOS and Displays on Any Type Terminal. Features: Four levels of play, Swap side, Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is **one of the strongest CHESS programs running on any microcomputer**, *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*
F, S and CCF - $79.95

There are no complications for the decoding of L1, as our first row (with the help of phi-7) covers all available minterms, and results in the expression :

L1 = y2

The decoding goes like this. Place 'x' under minterm 1, and write 001 to the right. "Is 1 + 4 = 5 available? Yes!" So a 4 is entered below minterm-5, and bit-4 to the right changed to a phi. "Is 1 + 2 = 3 available? Yes!" But before we can mark it off we must first check whether 5 + 2 = 7 also is available. Minterm-7 happens to be a phi, which we can use if we choose, so we place a 2 under both minterms 3 and 7 and change bit-2 to a phi. Finally "Is 1 - 1 = 0 available? No!" So our decoding is complete. We don't start a new row for minterm-4, as it's a phi, and we don't HAVE to use it.

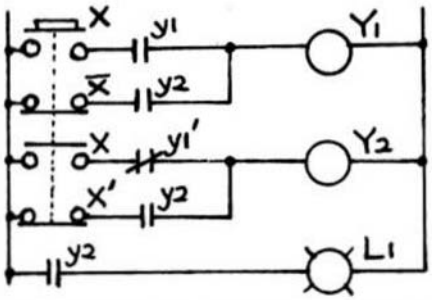


Diagram 23

The complete network is shown in Diagram 23, with the networks for each device being kept separate for now. As individual networks, we've produced the minimum expression for each, but later on, we'll learn how to "roll up" the separate networks into a combined network in order to save even more relay contacts. You'll observe that as the expression for L1 is simply L1 = y2, we could save a single contact by merely connencting L1 across the coil of Y2, so that each time Y2 became energised so also would L1. However, if L1 were a heavy-current device, and X had only light-duty contacts, we would leave L1 to be controlled by Y2.

HAZARDS

Up to this point we've assumed that push-buttons have snap-action, such that they operate more or less instantaneously on being pressed or released. On this assumption, the networks shown in Diagram 23 are quite valid. Taking Y2's network as an example, we see that initially Y2 is de-energised, but if we operate X then Y2 becomes energised (via X and y1'), thus closing its own contact, y2, in its lower path. If X is now released, and has snap-action, Y2 will have its current sustained through this lower path, even though the upper path is now open-circuited. BUT, if X does NOT have snap-action, and it's pressed as before, and y2 again closes in the lower path, then, if X is released very slowly, it's possible to enter an intermediate stage where the X upper-contact has opened but the lower X' contact has not yet closed. Under these conditions, Y2 may have time to "drop out" instead of staying in as per specs.

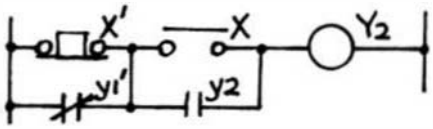


Diagram 24

This condition is known as a "hazard", and is recognised from the decoding-table by the fact that the two rows of the decoding have a valid transition between them, namely a change from X' to X, and there is also no "overlap" between minterms in the decoding-rows, one row covering minterms 1 and 3, and the other minterms 4 and 5. The solution in such a case is to try decoding 0s instead of 1s. That is, to draw up a new decoding table for the decimal numbers 0, 2, 6 and 7, which are now referred to as maxterms (instead of minterms) because we'll be complementing the decodings for each device.

For example, if we were decoding 1s, the minterm-1 (binary 001) would be read as X'y1'y2, but the MAXTERM-1 (binary 001) would be read as X + y1 + y2'. Everything is read out in reverse when decoding 0s and reading maxterms. Basically, we're saying that there IS a 0 in the X-position, plus there IS a 0 in the y1-position, but there's NOT a 0 in the y2-position. Diagram 24 shows a hazard-free network derived from decoding 0s instead of 1s. Keep in mind that if there are any phis in the minterm decoding-table, they should also be retained in the maxterm decoding-table, as a phi may be used as either a 1 or a 0 in order to obtain an optimum decoding. Sometimes it happens that in the process of eliminating a hazard from the minterms, a new one gets created in the maxterms. In such a case, in order to ensure maximum reliability of operation of our network, we'd be forced (see Diagram 22 for Y2) to use the non-essential row-1 as well to provide the necessary overlap between the other two rows.

Hazards should ALWAYS be looked for, AND ELIMINATED, in this way. It makes for a more reliable circuit even if the push-button, for that matter, IS a fast-operating device. There's no guarantee that it won't get a little sluggish as the years roll by!!

IS THAT UNCLE FRED WE SEE AHEAD? SURELY NOT!

This is kind of embarrassing for me, as I recall telling you at the start of our journey that I haven't lost a single soul so far in all my expeditions. Truth WILL out sooner or later, I guess, and Uncle Fred has put me in the position of having to confess to you - I lied! Yes, I lied, but it was only to protect you from being overly nervous during our journey. I didn't want you jumping out of your skins every time a little twig snapped, or a bit or byte fell to the ground. Now, however, I'm forced to admit that on my previous trip, somehow or other, I "mislaid" Uncle Fred in this area. Naturally, I assumed that the M'bul-yans had got him and we'd never see him again. Having got that off my chest, let's go find out out what happened, shall we? I hope you all forgive me, as my motives for lying were well-meant!

Well! Hello there, Uncle Fred! Where have you been all this time? We were getting a little anxious when you got separated from us! Haven't slept a wink since then, worrying about you!

UNCLE FRED'S STORY

(Now Uncle Fred's speaking) I believe you, though no-one else would! Actually, I've been through a most trying time. I'd fallen behind the rest of the gang, as I spied an interesting bit-position just off the trail, when BINGO, next thing I knew I was gagged, tied hand and foot, and slung on a long bamboo pole. Two guys then trotted off into the jungle, with me hanging from this pole slung between them, just like a pig at a barbecue roast. After a while we got to their village, where I met their medicine-man/chief, name of I-asku. He spoke English fairly well, and explained to me what was going to happen to me. Seems like they have this custom, going way, way, w-a-y back into their past, for disposing of prisoners. Rule 1 is that ALL prisoners MUST be taken alive, and taken good care of. These M'bul-yans aren't cannibals, but apparently their ancestors are, hence Rule 2. Rule 2 says that no matter how many prisoners they have "in stock", one of them is to be selected at random each day at sun-rise, and invited to make a statement to I-asku. Any statement, in any language whatsoever!

Then, during the day, I-asku would examine this statement to decide if it's TRUE or FALSE. It doesn't matter whether it's ACTUALLY true or false - only his learned opinion on the matter counts. So if you said that men had been to the moon, and he thought that couldn't possibly be so, then it was NOT so! Then, at sun-down, if he decided the statement was TRUE, the prisoner got thrown to the sacred crocodiles, so that when his soul ascended to the Great Big Never-Never Land, the M'bul-yan's ancestors would grab it and eat it. On the other hand, if the statement were judged false, the soul had to be first purified by fire, and so the prisoner got burned alive at the stake. Also, if one made statements such as "Michaelangelo had a wart on his rear end!" or "There'll be a major war in the year 2020", hoping to delay your death while I-asku does research on Michaelangelo, or until 2020 rolls round, the prisoner was obviously a smart-aleck, impure of heart, which necessitated purification by fire before the ancestors could enjoy their little feast. Statements in languages foreign to I-asku were also classed as "impure", as such were obviously meant to make his analysis difficult, to say the least.

Then they threw me into a compound where I stayed for a few days, listening to the others being taken out, one each day, followed at night by either a splash and a quick, short scream, or the crackling of flames and long-drawn-out groans going on for the longest while. In the end, everyone decided they would rather have a quick death, and planned statements such as "Grass is green", "2 + 2 = 4", "You nice guy, I-asku", and so on. Same with me! Then one day, I remembered your saying that if only one could map a logic-problem, the problem was automatically solved at the same time. I didn't really see how this could help me, but I scratched a little K-map in the dust, and entered 1s and 0s to describe my situation. And sure enough! There was the answer staring me in the face.

Just by chance, they selected me the next morning, so when I-asku stood in front of me, he said "Ah, you famous Uncle Fred! You mebbe happy know Aunt Minnie also happy inherit you 95 dollar. Now she have 194 dollar. Make statement please!" So I made the statement extracted from the K-map. At first he didn't say much. Then he went into a sort of spirttual dance, asking for guidance from his ancestors. Then he came back to me, and said "I, I-asku, I tell you." - (This I-asku has quite a sense of humour!) - "You one smart cookie, Uncle Fred!" and asked if I'd prefer to make a different statement perhaps. He explained that this situation had never, ever, occurred before (and as he was forbidden by the rules to kill me in any other way, otherwise terrible disaster would fall on the tribe) he was in a tight spot. Would I like to reconsider? Otherwise he'd have no alternative but to return me to where I'd got "picked up", and then set me free. Naturally, I declined his kind offer to reconsider, so at sundown a few days ago they dropped me off back here on the trail. And here I've been ever since, waiting for you to turn up.

I guess you're all anxious to know what I told I-asku. Well .....

(Interruption by me!) Hold on there, Uncle Fred! Don't make it too easy for my gang. Let them think it over for a while themselves, and try to figure out what THEY would have said. Better still, try to figure it out from a K-map. How about it, fellas? Uncle Fred will give the solution later. Summarising, only two forms of death allowed - crocodiles if the satement is true (in I-asku's opinion), otherwise burned at the stake if false, undecipherable, sneaky, smart-alecky, or otherwise "impure of heart". Yet Uncle Fred figured out how to avoid either death. What did he tell I-asku? And how did he figure it out?

When you've sorted THAT out, we'll proceed to

TEST SEVEN

Design the following circuits :

1. Light L1 to come ON when X is pressed, and go OFF when X released.

2. As above, except that the current for L1 must NOT flow through X. This means that any horizontal movement through the flow-table must NOT produce a change in the state of L1. Such a change can only occur via Relay-action, that is, when movement takes place vertically in the table.

3. L1 to come ON when X is pressed, and stay ON thereafter, even when X is released.

4. L1 is OFF initially. When X is pressed it stays OFF, but comes ON when X is released, thereafter going OFF each time X is pressed, and coming ON again each time it's released.

5. Two lights, L1 and L2. L1 to come ON when X is pressed, and to go OFF when X is released. When X is pressed again, L2 is to come ON, and go OFF when X is released. Subsequent operations of X to repeat this pattern of alternating L1 and L2.

6. As in 5 above, except that after X is pressed and released for the second time, subsequent operations of X merely repeat that of the second. That is, L2 comes ON each time X is pressed, and goes OFF each time X is released.

7. L1 to come ON when X is pressed the first time, and to stay ON thereafter, L2 to come ON when X is operated for the second time, and stay ON thereafter.

8. Lights initially OFF. When X is first pressed, L1 is to flash ON and OFF continuously. If X is released at the moment when L1 is OFF, the circuit is to return to its initial state, but if L1 happens to be ON, the circuit is to switch to a state where L1 goes OFF, but if X is now pressed again L2 will flash ON and OFF continuously. Now, if X is released when L2 is OFF the circuit goes back to its PREVIOUS state, ie, where L1 and L2 are OFF, but waiting for L2 to flash if X is pressed again. However, if X is released at the moment L2 is ON the circuit is to return to its INITIAL state. HINT : When two-way action occurs between two boxes in a State-Diagram, the line connecting these two boxes has an arrow-head at both ends to indicate this.

9. As in 8 above, except that if X is released when L2 happens to be ON, it stays ON, and resumes flashing when X is re-pressed.

10. As in 8, except that if L2 is ON when X is released, it goes OFF and L1 comes ON. L1 goes OFF and L2 resumes flashing if X is re-operated.

Yep, I know. This is quite a pile of problems, but don't forget - you didn't have any at all at Mile 6. Some may appear rather frightening when you first read them, but if you just take your time you'll find that as you enter the action on the flow-table, they're not so bad after all. Don't worry if you can't do them all. Just take it steady, and try again in a few days time. Even though you can probably draw the low-numbered circuits directly from the specs, I'd still like you to go through the whole design process. This will not only give you lots of practice, but will also serve to satisfy you that the circuits do, in fact, turn out the way you would otherwise have designed them.

So here we are at Mile 7, looking down the road towards Mile 8. During this next stage, we'll refine our design technique somewhat with a more complicated example. Just think! Who'd have thought a few short miles back that we'd be designing tricky little circuits like those above quite so soon? And the higher-numbered problems ARE tricky - even for seasoned seat-of-the-pants designers. With our admittedly unrefined technique to date, however, they should fairly easily come together to produce a workable circuit the very first time.

So, I'm just going off to one side for a little chat with Uncle Fred, as we're past M'bul-yan territory now, and quite safe. At least I don't have to lie any more about not losing anyone, though I bet Aunt Minnie won't enjoy having to return Uncle Fred's 95 dollars - unless she's spent it already! See you later.

EOF

FORTH ***

A Tutorial Series

By: R. D. Lurie
9 Linda Street

CALLING ROM ROUTINES FROM FORTH

Often, it would be very convenient to use ROM code as a subroutine called by a FORTH program. However, it is not always so obvious just how one should go about doing it! In this installment, I want to illustrate one way to do just that.

The main point to verify when considering calling a ROM routine is to make sure that it will not cause a problem with FORTH. Not only do you have to make sure that the routine does not do anything harmful to the RAM occupied by FORTH, but also that it does not permanently change any of the registers used by FORTH as pointers. All of the 680x FORTH's that I know of allow free use of the X register and the A and B accumulators, but the Y, U, and S registers must be restored when returning to FORTH. This really means that a routine written for the 6800 would probably not cause any problems, but one written for the 6809 might use the Y or U registers without restoring them, and this would be disastrous.

If you must use Y or U, I suggest the following outline for the calling definition:

```
CALLER  LDA   1,U     load into A from lsb of Data
Stack
              PSHS  Y,U     save Y and U
              JSR   ROM     jump to ROM routine
              PULS  Y,U     recover Y and U
              LEAU  2,U     the DROP operation
```

Figure 1. A suggested outline, in standard Motorola infix notation, for calling a ROM routine.

You can modify this as necessary to fit your requirements. The definition in Screen #2 is in the postfix notation normally required by a FORTH assembler.

A SPECIFIC EXAMPLE

First, some background—many people think that the use of hardware math ic's is a relatively new idea, just invented by Intel and/or IBM. Of course, like many other things that "everybody knows", that is just not true. I have one of the CALC- 1 S-30 bus math boards which was produced by SWTP nearly 10 years ago, and I am still using it! I got it originally for use with 6800 Assembly language programs. The CALC-1 uses a National Semiconductor MM57109 calculator chip, which they had developed to go with their COPS system. I suspect that the MM57109 is no longer available, except on the surplus market. It is rather slow when compared to the more recent math ic's, since it works at about 400-500 KHz. However, it is fast enough for most hobby applications; although, I would expect speed problems with calculations for animated graphics.

About the only important fault of the CALC-1 was that I could never find a direct way to enter negative exponents. There may be a way to do so, and I would appreciate any information that anyone may have on this. I enter the negative exponent as a 5- step operation:

1. Enter the exponent as 1 to the positive exponent value.
2. Take the reciprocal (1/X).
3. Enter the number as a positive, floating-point value.
4. Multiply the resulting two numbers.
5. Change the sign, if necessary.

This is not very elegant, but it does work.

At this point, you are probably wondering why I am wasting your time talking about an obsolete math board. The reason is that even though the math board may be obsolete, the techniques for talking to it are not!

THE ROM ROUTINE

I should explain at this time that I am using a GMX 6809+ CPU which has RAM available at $E700, so I often use this address page for development of software which I later expect to transfer to ROM. I also use this page for scratchpad RAM, as necessary. I have installed the MCALC-1 software at $E71C, with its required RAM starting at $E7F1. I will put the MCALC-1 software, which I call FCALC, into EPROM when I am finally satisfied with it. I now load the $E700 block of software with my FLEX STARTUP.TXT file.

The FCALC listing shown is the version I presently use, except for the first two lines at the lable, "CALC0". Since not everyone will have the $E700 page of RAM available, I have included the ROM listing in the form necessary for use in the RAM just below FLEX. As you can see, I have set the ORG lable to $B800 to allow plenty of room for any modifications you may wish to make. I have written the ROM routine so that it can be called from any program, not just one written in FORTH. The program is position- independent, except for the external references. Notice that I have used extended addressing for the CALC-1 addresses. This could present a problem if I ever want to move the CALC-1 board, but I don't know why I might do that, so I have not worried about it. The RAM references have to be to fixed addresses simply so that they can be referenced by the calling programs. If you use this routine, and later transfer it to ROM, be sure that you leave the variables and buffer in RAM!

If you want to use this routine with a Pascal program, you might have some difficulty with accessing the RAM addresses, because of the problems inherent in Pascal, but I will leave that up to anyone who might be that interested in a particular application.

The only register I used for indexed addressing is the X register, because that is normally the only one free in 6809 FORTH's. You will have to make a few minor changes in the routine for use with the 6800, since I took advantage of the D accumulator on several occasions; but splitting the D into separate A and B is all that is required.

THE FORTH PROGRAM

The actual FORTH program which I use is shown in screens #2-#10. Screen #1 is simply the loading screen, and screen #12 is a diagnostic and debugging aid. You probably will only want to load SHOW while you are developing a program, but not for the final application.

Screen #11 is necessary only for testing FCALC and using it as a desk-calculator. DATA simply provides a way to enter keyboard data, and, normally, is not used when calling FCALC from a FORTH program. Later, I will show what I think is a much better way to enter a large quantity of data into a number-crunching program.

Line #2 of screen #11 simply creates the entry buffer and provides the necessary starting address for use by @INPUT . There is no real need for an 80-character buffer, but I have let that become a habit. One of these days, it will probably get me into trouble, but such a large buffer does no harm if there is enough RAM to allow it.

Lines #5-6 simply clear PAD1 and accept up to 80 characters, or less, if terminated by the usual <CR>. Note that there is no error protection, here, as DATA is not intended to be a permanent part of a program. If you want to use FCALC as a desk-calculator, you don't expect much error protection, anyway; but go ahead and add it, here, if you want it.

Line #7 puts the address of PAD1 on the Data Stack for use by @INPUT and then calls that word for the actual data entry.

Line #8 is necessary in order to tidy the Data Stack, since @INPUT returns the address of (the last digit entered + 1). This address is not used by DATA, so it must be removed from the Data Stack.

The real meat of the program starts with @INPUT in screen #10, which is the primary point of data entry.

@INPUT must be entered with the address of the input ASCII string already on the Data Stack. By doing this, we can use any source we wish for the input without having to worry about putting all kinds of error protection, etc. into @INPUT . As we go along, processing the input string, we will be maintaining a count on the Data Stack. This count is initialized to 0 in line #1.

The data string is read by means of a BEGIN ... UNTIL loop containing an IF ... ELSE ... THEN conditional construct which does the work of the definition and also sets the flag used to escape the loop after the last character has been read.

In line #3, 2DUP is used to make a copy of the starting address of the string and the current value of the counter. The sum of these two numbers acts as a pointer to the next character to be read. Since the string has the form of a "packed string", that is, one byte per character, the C@ byte-read form must be used to fetch the character to the Data Stack.

The character now on top of the Data Stack serves two potential functions; it is used as a flag to determine the end of the string, and it is used as the input value if it is not the input string terminator. Therefore, in line #4, DUP is used to make a copy of the character for use as the flag. The character is first compared to BL ($32). OVER is used to make a second copy of the character, without disturbing the Boolean result of the first comparison. The copy is now compared to 0, and OR is used to "add" the two Boolean flags. It is this sum which becomes the flag for the IF ... ELSE ... THEN conditional. By using the Boolean sum of the comparison with BL and 0, we can clear the input buffer either with BLANK (BLANKS in FIG-FORTH) or with ERASE . This makes the @INPUT definition much more general.

The flag generated by the OR is TRUE (non-zero) if the last character obtained from the string was either a space or a zero ($20 or $00). Any thing else would be interpreted, at this point, as a valid character.

If the flag were TRUE , then the IF part of the conditional would be executed. In that situation, line #5 would DROP the extra copy of the character from the Data Stack. The current value of the count would then be added to the original string pointer and the resulting sum would be incremented by one (1+) to point past the terminating character. The purpose of this will be discussed later.

Line #7 then places a TRUE flag on top of the Data Stack in order to force the end of the BEGIN ... UNTIL loop.

If the flag generated in line #4 had been FALSE , then the ELSE portion of the conditional would have been executed. The first action would have been a jump to the SCREEN-INPUT definition, which processes correct input, or signals an error situation and causes an ABORT upon receipt of invalid data. If the data could be processed properly by SCREEN-INPUT , then the count on top of the Data Stack is incremented by one (1+), but not added to the original address at this time. Line #10 then places a FALSE flag on the top of the Data Stack in order to force the BEGIN ... UNTIL loop to search for the next character of the input string.

Once the entire string has been processed, lines #2-12, the ENTER definition is called (screen #4) in line #13.

The last action of @INPUT is to set the sign of the entry, based upon the value stored in _MINUS-FLAG . _MINUS-FLAG will have been conditioned by SCREEN-INPUT , so that @INPUT only has to read its value in line #14, and use that value as the BOOLEAN flag for line #15. If the flag had been TRUE (non-zero), then CHANGE-SIGN (screen #4) would be executed and _MINUS-FLAG would be set FALSE (0), in preparation for the next use of @INPUT .

SCREEN-INPUT is one of those classic situations just crying for the CASE ... ENDCASE structure. Its purpose is to screen the incomming data from @INPUT and take appropriate action, based on the characteristics of each item of input. The first action of SCREEN-INPUT is DUP , because the CASE ... ENDCASE structure executes a DROP of the input character, which we cannot prevent, and we need to use this character in lines #3 and #5.

Line #3 checks the input character for an ASCII value ranging within the values of $30 and $39. If the character has this value, the ASCII value for 0 ($30) is subtracted from it, and CALC (screen #3) is called. CALC is the definition which calls the ROM routine.

A decimal point is processed in line #4. The DROP eliminates the excess copy on the Data Stack and then calls DECIMAL-POINT (screen #4), which does the actual work.

Line #5 processes a minus sign which indicates a negative number. This program cannot process negative exponents; refer to the previous discussion for the reason. In this context, the minus sign only denotes a negative number, and not a subtraction operation. The minus can be anywhere within the string, including between two digits, and it will still function to denote a negative number. I don't know if I have tried all of the possible combinations, but I think that one or more minus signs anywhere within the string will act as if only one minus sign had been positioned as the leading character of the string.

A plus sign has no real value in this context, so it is simply cast aside with a DROP in line #6. The plus sign can appear in the input without doing any harm, but it serves no actual function. However, it is sometimes convenient to use the plus sign as part of the input, and it might also be used accidentally, so we want to be sure that its innocous use would not cause the whole program to crash, unnecessarily!

The letter E, either upper or lower case, signals the beginning of the exponent. The DROP clears the Data Stack and ENTER-EXPONENT (screen #4) takes the appropriate action.

These are the only acceptable inputs, and any other character causes a fatal error. In line #9, the input error is signaled and ABORT closes down the operation.

At the risk of boring those of you who do not have a CALC-1 math board, I want to describe how I read the output, since the principles should apply to any other math output, even one based entirely on software.

Screens #6-8 process the output from the CALC-1 math board. These are simply a translation of the OUTCHR, etc. routines from the original SWTP 6800 machine code into a convenient FORTH structure.

The .ANSWER definition is a good example of factoring a definition in order to make it easier to read, write, and later understand. The word FORMAT in line #1 of screen #8 actually refers to a variable used by the ROM routine. Since it is a single-byte variable, it must be read with C@ . As I am sure you have noticed, I have adopted the convention of using _ (underline, $5F) as a prefix for any VARIABLE . However, I have not settled on just what to do with variables which are only one byte long and/or part of the system or ROM, rather than part of the FORTH program, proper. The _ mnemonically represents a space to be filled, so whenever I see a word with the _ prefix, I don't have to look it up to know that it is a variable.

FORMAT contains $FF if the CALC-1 output is in scientific notation or $00 if the output is in floating point. Therefore, the simple IF ... ELSE ... THEN construct is all that is needed by this definition.

FP-NOTATION takes its input from the string of data in the *MCALC buffer. The * is another prefix which I have recently started to use; in this case, I use it to denote some sort of array. As with FORMAT , *MCALC is actually filled by the ROM program and not the FORTH program. *MCALC corresponds to BUFFER at $E7F4. I confess that I am not completely consistant with using my own naming conventions, but I expect to work that out as I go along.

The first action of FP-NOTATION is to read read the first byte of output and determine if it is a positive or negative number. The number is negative if the fourth bit is set; therefore, the phrase 8 AND will produce a TRUE for negative numbers and a FALSE for positive numbers. Line #1 makes this check and line #2 displays the minus, if appropriate.

The second byte contains the location of the decimal point, although it is not as easy to read as the sign of the number. Line #3 first reads the byte and then performs the Boolean AND with $0F in order to remove the upper four bits. The resulting number is then subtracted from $0D by the phrase 13 SWAP - . The result is the position of the decimal point, counting from left to right; this value is stored in _DP-POS .

Since the output from CALC-1 is a maximum of eight digits, plus the decimal, the digits may be found in the second through the tenth position of the *MCALC array. The digits are stored as conventional ASCII, so they may be displayed as they are read. The only catch is in showing the decimal point in the proper location. I thought that the easiest way was with a DO ... LOOP . By setting the loop limits at 2 and 10, I could simply add the loop counter I (line #5) to the start of the array and immediately have a pointer to the current digit. Furthermore, by comparing I to the contents of _DP-POS , I would know when to display the decimal point (lines #5-6).

SCI-NOTATION is more complicated than FP-NOTATION, because we also have to be concerned about the exponent, both its magnitude and its sign. As it happens, the significant byte locations are also more scattered for scientific notation than for floating point notation, so we must skip around a little in order to be able to read the output. This is no big deal, it just confuses the eye a little when one first tries to understand the SCI-NOTATION definition.

As with FP-NOTATION, the first thing to display is the sign of the number, which is done in lines #1-2 of screen #6. This time, the sign is stored in the fourth bit of the third byte of *MCALC , and its recovery is analogous to FP-NOTATION .

The number in scientific notation is usually represented as a single digit, a decimal point, and as many following significant digits as appropriate. Therefore, since the first digit is to be found in the fifth byte of *MCALC , we only need to display it, and follow it, automatically, by a decimal point (lines #3-4).

Since there are normally seven digits following the decimal point, a DO ... LOOP , indexed by 5 and 12 can be used to display them (lines #5-7). I chose a more general way to fetch the digits, this time, as compared to the FP-NOTATION definition. Here, I have masked the fetched byte with $0F and then added the result to the value of ASCII 0. This was then used as the argument for EMIT . Actually, the form used in FP-NOTATION is completely adequate, and can be used here, with a concomitant saving in RAM and time; I just used one form in each definition to illustrate a point.

Line #8 displays the "E" required to signify the exponent to base 10. I included a space ahead of the "E" in order to make the output easier to read.

The sign of the exponent must be printed next, and it is found in the first bit of the third byte of *MCALC . Lines #9-10 read and display it. The phrase 1 AND isolates the bit, which is then used as the Boolean flag for the following IF ... THEN conditional.

The magnitude of the exponent is read and displayed by lines #14- 15 as taken from the first and second bytes of the *MCALC buffer.

Screens #4 and #5 contain the commands which are used to control the MM57109 calculator. The format was chosen simply to make the screens easier to read. If you have one of the CALC-1 boards, you should have no trouble figuring out how to use all of the definitions.

Those of you who don't have one, will only be interested in the algorithm. I will make the explanation a little easier to follow by treating the case of a desk calculator. Remember, the MM57109 uses RPN input, so it is much easier to program than some other beasties. Suppose you want to take the square root of the sum of 37.123 and .0051. The following command sequence would be used:

```
DATA<CR> 37.123<CR> ok
DATA<CR> .0051<CR> ok
PLUS<SP>SQUARE-ROOT<SP>.ANSWER<CR> 6.0932832
   ok
```

As soon as <CR> or <SP> is pressed, the word CALC in screen #3 is called. The number preceeding CALC in each of the definitions in screens #4-5 is the proper command to the MM57109, and it is placed on the top of the Data Stack just prior to calling CALC .

CALC is written in Assembly language, using the FORTH-style postfix notation. I just don't know of a way to jump to a ROM subroutine from high-level FORTH, unless your version has a word such as JSR . Since most don't, I assume that you will do as I have done.

The first step (line #1), is to move the least significant byte from the top 16-bit word on the Data Stack into Accumulator A. This is easy with most 6809 versions of FORTH, since the U register is usually used as the Data Stack pointer. If your FORTH uses a different pointer, I will have to leave that up to you; but the instruction manual should tell you where to find the required pointer.

Line #2 is the obvious extended jump to the ROM subroutine. Remember that any ROM address must be an absolute address, since you have no way of knowing the relative position of CALC within RAM from compilation to compilation.

In this particular application, you must remove the top word from the Data Stack before going to the next definition; otherwise, we would quickly fill the Data Stack with garbage. Therefore, line #3 simply executes the Assembly language form of DROP .

Screen #2 contains the three variables used by FORTH, but not by ROM. However, it also has two CONSTANT definitions, FORMAT and *MCALC , which act as pointers to a variable and an array used both by FORTH and by ROM. There are also three CONSTANT definitions which are pointers to the entry address for each of the three ROM routines, FCALC, OUTANS, and INITAL. These pointers were all defined as HEX simply for my convenience, FORTH couldn't care less!

The only time you need the INITAL pointer is when the routine is called the first time. INITAL initializes the 6820 PIA interface, and is never called after that. When I finally do put FCALC into ROM, I will either have to initialize the PIA interface at startup (the preferred action) or include the simple definition in Figure 2 as one of the first actions of my math initialization.

```
CODE  INITIALIZE-CALC-1
      INITAL    JSR
                NEXT,
 END-CODE
```

Figure 2. A method for initializing the CALC-1 interface directly from FORTH.

EASY DATA ENTRY

Earlier, I mentioned that I had found what I believe is an easy method for data entry into a number-crunching program. Screen #101 shows the idea; namely, simply use any convenient editor to enter the input data as an ASCII screen file, which is then read by the FORTH program as it asks for input.

Either line #0 can be used for a title, etc., or it can be used for data, if you need the space. Here, I show it used for a title, just to emphasize the point.

Whatever you decide to put on the first line, the first data item should be the count of the items to be read. In that way, you can read the rest of the input with a simple DO ... LOOP .

All you need to do to read the count is to use the phrase:

```
101 BLOCK 63 + NUMBER DROP
```

In this phrase, 101 BLOCK points to the start of the screen, and 63 + adds the necessary offset for the use of NUMBER . The DROP is needed, because NUMBER produces a 32-bit result, and we only want a 16-bit integer to use in a DO ... LOOP .

The actual data can be read from the block by following the outline of FORTH commands in Figure 3:

```
101  BLOCK  128  +      \ point to first digit
  ( count )  0  DO      \ reading loop limits
    ( adr )  @INPUT     \ read the input data
    ( appropriate calculator commands )
  LOOP
  DROP                  \ discard last copy of
                        \ pointer
```

Figure 3. Outline of the FORTH commands for reading input data from a "data screen".

If you refer to the definition of @INPUT (screen #10), you will find that I left an address on the Data Stack which was a pointer to the byte after the terminating byte. This becomes the address pointer to the next input for @INPUT , if you put only one space between consecutive numbers in your input screen. In other words, the first call to @INPUT must specify the starting address for the reading, but subsequent calls would not, because the address would already be on the Data Stack. Of course, you must enter the numbers onto the original "data screen" without any breaks, or you will have to respecify the starting address of the next byte of valid input.

If this explanation is not adequate for you to see my point, wait for the next installment, which will go into this in more detail, with specific examples.

SOURCES FOR FF9 AND FF2

If you still don't have FF9 or FF2, and I cannot imagine why you would not, you can still get a free copy. Send a stamped, self- addressed mailer with two disks to:

Wilson Federici
1208 NW Grant
Corvallis, OR 97330

Be sure to tell him whether you want FF9 for the 6809 or FF2 for the 6800. If you prefer, you can get FF9 from me (I don't have 6800 FLEX), but you will get quicker service from Wilson.

Come on, you can't find a better version of FORTH-83, and you sure can't beat the price.

```
SCR #1
  0 \ Loading screen
\ RDL 09/02/87
  1
  2 : MC
  3     2  10  THRU ;
  4
  5 CR  MC

SCR #2
  0 \ CONSTANTs, etc.
\ RDL 09/01/87
  1
  2 FORTH  DEFINITIONS
  3
  4     HEX
  5 E71C  CONSTANT  FCALC
  6 E7B4  CONSTANT  INITAL
  7 E7F1  CONSTANT  FORMAT
  8 E7F4  CONSTANT  *MCALC
  9     DECIMAL
 10
 11 VARIABLE  _DP-POS
 12 VARIABLE  _MINUS-FLAG

SCR #3
  0 CODE  CALC     ( b - )
\ RDL 09/01/87
  1     1  ,U    LDA                          \
fetch byte from stack
  2     FCALC    JSR
  3     2  ,U    LEAU                         \ DROP
input byte
  4              NEXT,
  5 END-CODE

SCR #4
  0 \ Control codes
\ RDL 09/01/87
  2 : DECIMAL-POINT    10  CALC ;    : ENTER-
EXPONENT    11  CALC ;
  3 : CHANGE-SIGN      12  CALC ;    : PI
13  CALC ;
  4 : SMDC             24  CALC ;    : X><M
27  CALC ;
  5 : >MEM             28  CALC ;    : MEM>
29  CALC ;
  6 : ARC              32  CALC ;    : ENTER
33  CALC ;
  7 : TOGGLE-MODE      34  CALC ;    : ROLL-STACK
35  CALC ;
  8 : SINE             36  CALC ;    : COSINE
37  CALC ;
  9 : TANGENT          38  CALC ;    : RAD>DEG
44  CALC ;
 10 : DEG>RAD          45  CALC ;    : CLEAR-CALC
47  CALC
 11                                      0
_MINUS-FLAG  ! ;
```

```
 12 : X><Y              48  CALC ;     : E**X
49  CALC ;

SCR #5
  0 \ Control codes
\ RDL 08/31/87
  2 : 10**X            50  CALC ;     : SQUARE
51  CALC ;
  3 : SQUARE-ROOT      52  CALC ;     : LN
53  CALC ;
  4 : LOG              54  CALC ;     : 1/X
55  CALC ;
  5 : Y**X             56  CALC ;     : PLUS
57  CALC ;
  6 : MINUS            58  CALC ;     : TIMES
59  CALC ;
  7 : DIVIDE           60  CALC ;     : POP
46  CALC ;

SCR #6
  0 : SCI-NOTATION      ( - )
\ RDL 08/27/87
  1     *MCALC  2+  C@      8  AND
  2       IF    ASCII  -  EMIT    THEN
  3     *MCALC  4  +  C@     15  AND     ASCII  0
+    EMIT
  4     ASCII  .  EMIT
  5     12  5  DO
  6       *MCALC  I  +  C@     15  AND     ASCII  0
+    EMIT
  7     LOOP
  8     ." E"
  9     *MCALC  2+  C@      1  AND
 10       IF    ASCII  -  EMIT    THEN
 11     *MCALC  C@     15  AND     ASCII  0  +
EMIT
 12     *MCALC  1+  C@     15  AND     ASCII  0  +
EMIT ;

SCR #7
  0 : FP-NOTATION ( - )
\ RDL 08/27/87
  1     *MCALC  C@      8  AND
  2       IF    ASCII  -  EMIT    THEN
  3     *MCALC  1+  C@     15  AND     13  SWAP  -
_DP-POS  !
  4     10  2  DO
  5       *MCALC  I  +  C@     EMIT
  6       _DP-POS  @     I  =
  7         IF    ASCII  .  EMIT    THEN
  8     LOOP ;

SCR #8
  0 : .ANSWER      ( - )
\ RDL 09/01/87
  1     FORMAT  C@
  2       IF    SCI-NOTATION
  3       ELSE  FP-NOTATION
  4       THEN ;

SCR #9
  0 : SCREEN-INPUT      ( c - )
\ RDL 09/01/87
  1     DUP
  2          CASE
  3   ASCII  0  ASCII  9  RNG-OF     ASCII  0  -
CALC      ENDOF
  4   ASCII  .  OF     DROP     DECIMAL-POINT
ENDOF
  5   ASCII  -  OF     _MINUS-FLAG  !
ENDOF
  6   ASCII  +  OF     DROP
ENDOF
  7   ASCII  E  OF     DROP     ENTER-EXPONENT
ENDOF
  8   ASCII  e  OF     DROP     ENTER-EXPONENT
ENDOF
  9     CR  ." ***INPUT ERROR***"  CR  ABORT
 10          ENDCASE ;

SCR #10
  0 : @INPUT ( adr1 - adr2 )
\ RDL 09/01/87
  1     0                                    \
initialize "count"
  2     BEGIN
  3       ( adr )  ( count )  2DUP  +     C@
  4       DUP     BL  =     OVER  0=     OR
  5         IF     DROP
  6           ( adr )  ( count )  +  1+
  7           TRUE
  8         ELSE  SCREEN-INPUT
  9           ( adr )  ( count )  1+
 10           FALSE
 11         THEN
 12     UNTIL
 13     ENTER
 14     _MINUS-FLAG  @
 15       IF     CHANGE-SIGN     0  _MINUS-FLAG  !
THEN ;

SCR #11
  0 \ PAD1  DATA
\ RDL 08/30/87
  1
  2 CREATE  PAD1  80  ALLOT
\ RDL 08/30/87
  3
  4 : DATA   ( - )
\ RDL 08/30/87
  5     PAD1  80  ERASE
  6     PAD1  80  EXPECT
  7     PAD1  @INPUT
  8     ( adr2 )  DROP ;

SCR #12
  0 : SHOW   ( - )
\ RDL 09/02/87
  1     CR  ." X= "     .ANSWER
  2     ROLL-STACK
  3     CR  ." Y= "     .ANSWER
  4     ROLL-STACK
  5     CR  ." Z= "     .ANSWER
  6     ROLL-STACK
  7     CR  ." T= "     .ANSWER
  8     ROLL-STACK
```

```
 9      X><M
10      CR  ." M= "     .ANSWER
11      X><M
12      CR ;

SCR #101
  0 \ DATA
\ RDL 09/01/87
  1 32
  2 5500 5500 5500 5500 5200 5200 5200 5200 5000
5000 5000 4800 4700
  3  5000 4600 4800 4500 4400 4700 4700 4600 4400
4600 4600 4500 450
  4 0 4500 4500 4700 4800 4800 4800

EOF


        ********************************************
                    * F-CALC    ver 1.0
08/26/87
        ********************************************

  C100                      ORG    $C100

  C100 8E   B7FF     CALC0  LDX    #$B7FF
  C103 BF   CC2B            STX    $CC2B     ME-
MEND
  C106 BD   B898            JSR    INITAL

  C109 7E   CD03            JMP    $CD03
WARMS

        ********************************************
                    *     EXTERNAL REFERENCES
        ********************************************

             E060  PORT60  EQU    $E060
             E061  PORT61  EQU    $E061
             E062  PORT62  EQU    $E062
             E063  PORT63  EQU    $E063

        ********************************************
                    *     ENTRANCE POINT FOR
SUBROUTINE VERSION
        ********************************************

  B800                      ORG    $B800

        ********************************************
                    *     PROCESS "ENTER"
        ********************************************

  B800 81   21       FCALC  CMPA   #$21
  B802 27   45              BEQ    ZERMEM

        ********************************************
                    *     MOVE "INSTRUCTION" TO
PORT
        ********************************************

  B804 F6   E061     OUTINS IDB    PORT61
  B807 2A   FB              BPL    OUTINS
  B809 B7   E060            STA    PORT60
  B80C F6   E060            LDB    PORT60
  B80F C6   3C              LDB    #$3C
  B811 F7   E061            STB    PORT61
  B814 F6   E061     OUTIN6 LDB    PORT61
  B817 2A   FB              BPL    OUTIN6
  B819 F6   E060            LDB    PORT60
  B81C C6   36              LDB    #$36
  B81E F7   E061            STB    PORT61

        ********************************************
                    *     SCREEN FOR "IMMEDIATE-
ACTION" COMMANDS
        ********************************************
                    *     "MASTER CLEAR"
        ********************************************

  B821 81   2F       MCLEAR CMPA   #$2F
  B823 26   03              BNE    MCLEA3
  B825 7F   B8A8            CLR    FORMAT
  B828 7D   B8A9     MCLEA3 TST    SMDC
  B82B 26   1C              BNE    ZERMEM

        ********************************************
                    *     "HALT" ( EQUIVALENT TO
NOP )
        ********************************************

  B82D 81   0F              CMPA   #$F
  B82F 27   66              BEQ    EXIT

        ********************************************
                    *     "SET MANTISSA DIGIT
COUNT"
        ********************************************

  B831 81   18              CMPA   #$18
  B833 26   05              BNE    INVER
  B835 73   B8A9            COM    SMDC
  B838 20   5D              BRA    EXIT

        ********************************************
                    *     "INVERSE MODE"
        ********************************************

  B83A 81   20       INVER  CMPA   #$20
  B83C 27   59              BEQ    EXIT

        ********************************************
                    *     "ENTER EXPONENT"
        ********************************************

  B83E 81   0B              CMPA   #$B
  B840 23   55              BLS    EXIT

        ********************************************
                    *     "TOGGLE MODE"
        ********************************************

  B842 81   22              CMPA   #$22
  B844 26   03              BNE    ZERMEM
  B846 73   B8A8            COM    FORMAT
  B849 7F   B8A9     ZERMEM CLR    SMDC
```

```
                ********************************************
                *       FILL BUFFER WITH
<SP>'S
                ********************************************

B84C 8E   B8AB    SETMEM  LDX     #BUFFER
B84F 6F   80              CLR     0,X+
B851 C6   20              LDB     #$20
B853 E7   80      SETME3  STB     0,X+
B855 8C   B8B7            CMPX    #BUFFER+12
B858 26   F9              BNE     SETME3

                ********************************************
                *       MOVE DATA FROM PORT TO
BUFFER
                ********************************************

B85A F6   E061    OUTANS  LDB     PORT61
B85D 2A   FB              BPL     OUTANS
B85F B6   E060            LDA     PORT60
B862 CC   163E            LDD     #$163E
B865 FD   E060            STD     PORT60
B868 F6   E061    OUTAN6  LDB     PORT61
B86B 2A   FB              BPL     OUTAN6
B86D F6   E060            LDB     PORT60
B870 86   0F              LDA     #$F
B872 B7   E060            STA     PORT60
B875 8E   B8AB            LDX     #BUFFER
B878 F6   E063    OUTA12  LDB     PORT63
B87B 2B   07              BMI     OUTA17
B87D F6   E061            LDB     PORT61
B880 2B   0D              BMI     OUTA22
B882 20   F4              BRA     OUTA12
B884 B6   E062    OUTA17  LDA     PORT62
B887 84   0F              ANDA    #$F
B889 8A   30              ORA     #$30
B88B A7   80              STA     0,X+
B88D 20   E9              BRA     OUTA12
B88F 86   36      OUTA22  LDA     #$36
B891 B7   E061            STA     PORT61
B894 B6   E060            LDA     PORT60
B897 39           EXIT    RTS

                ********************************************
                *       INITIALIZE INTERFACE
PIA
                ********************************************

B898 CC   7F36    INITAL  LDD     #$7F36
B89B FD   E060            STD     PORT60
B89E CC   0034            LDD     #$0034
B8A1 FD   E062            STD     PORT62
B8A4 B6   E062            LDA     PORT62
B8A7 39                   RTS

                ********************************************
                *       CONTROL FLAGS
                ********************************************

B8A8 00           FORMAT  FCB     0
B8A9 00           SMDC    FCB     0
B8AA 00           ERRFLG  FCB     0

                ********************************************
                *       OUTPUT BUFFER
                ********************************************

B8AB              BUFFER  RMB     12

                ********************************************

                        END     CALC0

0 ERROR(S) DETECTED

SYMBOL TABLE:

BUFFER B8AB   CALC0  CI00   ERRFLG B8AA   EXIT
B897   FCALC  B800
FORMAT B8A8   INITAL B898   INVER  B83A   MCLEA3
B828   MCLEAR B821
OUTA12 B878   OUTA17 B884   OUTA22 B88F   OUTAN6
B868   OUTANS B85A
OUTIN6 B814   OUTINS B804   PORT60 E060   PORT61
E061   PORT62 E062
PORT63 E063   SETME3 B853   SETMEM B84C   SMDC
B8A9   ZERMEM B849
```

68XX(X) And The STD BUS

By: Bill West

BILL WEST INCORPORATED
174 Robert Treat Drive
Milford, Connecticut 06460
203 878-9376

I have been promising to discuss the design of the STD-RIOC for several months now, and will finally do so in this article. Before jumping into the technical stuff, however, let me address several other points. The first is that the STD-RIOC is a product of my company, Bill West Incorporated, and this article should not be considered an impartial review or evaluation. The intent is to provide the reader with two things: a greater understanding of the STD bus by describing the design of a particular STD bus product, and an introduction to a product that may be of interest to some of you. The other point I want to mention is that I have very little idea of the technical background of the majority of 68MJ readers. The majority of the logic design of the STD-RIOC is implemented with PAL(tm) devices. For someone familiar with such devices, merely listing the logic equations might be sufficient exposition of the design, while other readers might desire a multi-part tutorial on digital logic circuits. The following presentation is somewhere between these extremes.

The first step in any design is figuring out what you are trying to do. In the case of the STD-RIOC, the goal is to be able to use STD bus I/O cards with computer systems not based on the STD bus. Specifically, we want to allow computer systems which use the Motorola Remote I/O Channel (VME bus systems and the GMX Micro-20) to use an STD bus card cage as an I/O expansion bus. In order to due this, we must in some way adapt the signals of the two buses to be compatible.

The signals of each of the two buses with which we are dealing can be broken into four groups. These include a power bus, a data bus, an address bus, and a control bus. The specific signals in each group are detailed in previous articles in this series (STD - Nov. 87, RIOC - Jan. 88). The power bus is the simplest to deal with. We will provide a separate power supply for the STD bus card cage and use the 50-pin option of the RIOC, which means the only "power" signal which must be connected between the two buses is logic ground. This is merely a direct connection between the two systems. The data bus is almost as easy. Both the RIOC and the STD bus use eight data lines. However the data lines must be buffered to insure that data is transmitted reliably between the two systems over the interconnecting cable. For you programmers out there, a buffer in a hardware system is not the same thing as a buffer in a software system. Programmers refer to a chunk of memory where data can be temporarily stored as a buffer. In a hardware design, a buffer provides no logical function, but merely provides the necessary voltage or current drive to transmit a signal, or the necessary signal conditioning to receive a signal. The logic of a signal is the same at both the input and output of a buffer. Since we are dealing with a bi-directional data bus that allows data to be transmitted either to or from the RIOC to the STD bus over the same eight wires, we must provide a data buffer that can be "turned around". We will use an 74LS245, which provides the eight buffers needed, along with "direction" and "enable" control lines.

Connecting the twelve-bit address bus of the RIOC to the STD bus is in some sense easier than connecting the data bus. The address bus is unidirectional, with address information always originating from the RIOC and being received by the STD bus. Since we are primarily interested in using I/O cards on the STD bus, and since the largest number of I/O cards on the STD bus are Z80 compatible, we will consider how to connect the twelve address lines from the RIOC to the Z80 STD bus. The Z80 STD bus (and the other 8 bit versions of the STD bus) support a 16-bit memory address bus and an 8-bit I/O address bus, with a memory expansion (MEMEX) and an I/O expansion (IOEX) line available. Again, since we are interested in using I/O cards, we will consider how to map the 12 address lines of the RIOC to the eight address lines of the STD I/O bus. The obvious thing to do is to connect the eight low order address lines of the RIOC (A0 - A7) to the corresponding lines of the STD bus. The next two address lines are used to select between four pages (256 bytes each) of STD bus memory. These four pages include the regular and expanded I/O pages, 256 bytes of memory mapped I/O, and one page for addressing control registers on the STD-RIOC. The remaining two address lines are used as select lines for up to four STD-RIOC interface boards, allowing a single RIOC to support four STD expansion card cages.

The fourth group of signals that must be dealt with are those that comprise the control bus. The control bus includes the STD MEMEX and IOEX signals mentioned in the previous paragraph, along with the lines that provide timing information to the bus and allow the servicing of interrupts. First let's consider the simplest form of read and write data transfers for a generic STD bus card, as described in the STD Bus Specification and Practice guide from the STD Manufacturer's Group (STDMG). The first step in a data transfer on the STD bus is selecting the desired memory or I/O address. Depending on the specific type of transfer, this will require activating from eight to sixteen address lines, along with either MEMRQ* or IORQ*, and possibly either MEMEX or IOEX. All of these lines are considered address qualifiers and may be activated in any sequence. After address setup time requirements have been met, either RD* or WR* is activated. Typically, in a write cycle the bus master will place data on the data bus before activating WR*, but this is not required. Data is latched on the rising (trailing) edge of RD* or WR*, and is held on the bus for the required hold time. The address selection lines remain valid until after the rising edge of either WR* or RD*.

Now let's consider the signals available from the RIOC. There are 12 address lines, a write line (WT*), and the data strobe (STB*), along with the XACK* input, which is used to terminate a data transfer cycle. The RIOC master sets up the address lines and the WT* control line, and then brings STB* low. In a write cycle, the RIOC master also puts valid data on the data bus before activating STB*. When the slave has either transferred the data off the data bus in a write cycle, or placed valid data on the data bus for a read cycle, it activates XACK* to indicate to the RIOC master that it can end the current transfer. If a read cycle is taking place the master then transfers the data from the data bus. To end the cycle, the master raises STB*, after which the slave raises XACK*.

How do we use the signals available from the RIOC to control the STD bus? The data bus and address bus we discussed above, and except for some mapping of the address bus the signals on the two buses are compatible. Let's assume we wish to address an I/O device on the STD bus, and also that we are not using the I/O expansion addresses. This means that the IOEX line must be low, and so must the IORQ* line. Since these are considered address qualifiers on the STD bus and do not contain any timing information, if we only want to do I/O transfers in the regular (non-expanded) I/O page, these lines can be set low permanently. We need to generate RD* or WR* for a read or write cycle respectively. We do this by combining the RIOC signals STB* and WT* according to the following equations:

RD* = (WT*'.STB*')' WR* = (WT*.STB*')'

The above equations use the same conventions Bob Jones is using in his column entitled "Logically Speaking" with one addition. I am using the "*" symbol to indicate a signal that is active low on the STD or RIOC bus. This means that logically speaking (sorry Bob) WT*' is the same as WT.

However, since the actual signal available to us is WT* (i.e. the voltage on the WT* line is near 0 volts during a write cycle, and near five volts during a read cycle) it is useful to explicitly state this in the equations. To state this in terms of hardware, the WT* signal is directly available from the RIOC, while to get a signal equivalent to WT, we need to add an inverter circuit. The inverter is represented by the "'" in the expression WT*'.

What the above two equations indicate is that whenever the RIOC signal STB* becomes active (goes low), either WR* or RD* on the STD bus will also become active (go low), depending on the state of the RIOC signal WT*. This is exactly what we need to start the bus cycle on the STD bus. How do we end the cycle ? The STD bus expects the master to terminate the cycle by raising either RD* or WR*, while the RIOC expects the slave to bring XACK* low to indicate that it has transferred data to or from the data bus. Apparently the STD-RIOC interface card will have to generate the appropriate signals for both buses, since it functions as a slave on the RIOC and a master on the STD bus. This implies that timing information will have to be generated on the STD-RIOC card. Leaving the timing details for later, this means that the STD-RIOC will generate an XACK* signal for the RIOC master after an appropriate delay from the falling edge of the STB* signal. When the STB* signal from the master goes high, it will force either RD* or WR* high, ending the STD bus cycle. (Refer to the two equations above.) The STD-RIOC will than bring XACK* high to complete the entire data transfer.

Although I have glossed over some of the details, the timing sequence described above is adequate for interfacing to static memories and some I/O devices on the STD bus. Many STD bus I/O cards, particularly those using 6800 type peripheral chips, require a higher degree of synchronization than that provided by the above timing. Let's consider a 6809 processor running at a clock speed of 1 MHz. It has two clock outputs, E and Q, which run at the same frequency but out of phase by 90 degrees. Each bus cycle starts with the falling edge of the E clock and ends with the next falling edge of the E clock. Each cycle is divided into four phases by the E and Q clocks. If we label the phases p0 - p3 (small letters indicating these are not signal names), the logic equations for the four phases are:

p0 = E'Q' p1 = E'Q p2 = EQ p3 = EQ'

Roughly speaking, address information is valid from the start of p1 to the end of the cycle, data being written by the 6809 is valid during p2 and p3, and data being read by the 6809 must be valid during p3. What does this mean if we use the 6809 on the STD bus ? It means we cannot just go in and give RD* and WR* strobes at random times (asynchronously) but must assert the signals in coordination with the E clock (synchronously). Since the STD bus only requires that the data be valid on the rising (trailing) edge of the RD* or WR* signal, a simple way to do this is to combine the E clock with the read/write signal of the 6809 (call it R) like this:

RD* = (ER)' WR* = (ER')'

This will let the 6809 generate a reasonable approximation of Z80 RD* and WR* signals.

The STD-RIOC is not equiped with a 6809 however, and it is necessary to generate timing information similar to that provided by the 6809 E and Q clocks by some other means. This is done with a standard microprocessor clock oscillator circuit and a 74LS163 4-bit binary counter, which is used to generate a sequence of sixteen states, which I will call t0 - t15. (I am using a small t to make it clear that these aren't signal names.) These t states can be considered a shorthand representation for the outputs of the counter. Let's call the counter outputs X0 - X3, with X0 being the least significant bit. The following chart shows the logic representation and the voltage levels of the counter output for each t state as it occurs in sequence. A 1 is used to represent a high voltage level, and a 0 a low voltage level.

t0	X3'X2'X1'X0'	0000
t1	X3'X2'X1'X0	0001
t2	X3'X2'X1X0'	0010
t3	X3'X2'X1X0	0011
t4	X3'X2X1'X0'	0100
t5	X3'X2X1'X0	0101
t6	X3'X2X1X0'	0110
t7	X3'X2X1X0	0111
t8	X3X2'X1'X0'	1000
t9	X3X2'X1'X0	1001
t10	X3X2'X1X0'	1010
t11	X3X2'X1X0	1011
t12	X3X2X1'X0'	1100
t13	X3X2X1'X0	1101
t14	X3X2X1X0'	1110
t15	X3X2X1X0	1111

Note that although this is a binary counting sequence and not a Gray code (refer to Bob Jones' articles again), that all the odd numbered t states differ from the preceding state by only one bit, X0. This means that if we only make things happen during the odd numbered t states we won't have to worry too much about things like slightly different transition delays on the counter outputs. I said we will use the t states as a shorthand for representing the logic we need to implement. The following two equations are an example of this, where F is some arbitrary function that I am using so that there is something on the left side of the equation.

F = (STB*)'(WT*)'X3'X2X1'X0 F = (STB*)'(WT*)'t5

The above equations both indicate that F is true when the STB* and WT* signals are low and the outputs of the binary counterequal 5, but the second is certainly easier to read. Now we can simulate the clock outputs of the 6809 like this:

E = t9+t10+t11+t12+t13+t14+t15+t0
Q = t5+t6+t7+t8+t9+t10+t11+t12

The reason I didn't start with t8 for the E clock is that as I mentioned above, going from an even to an odd numbered t state only one of the clock bits, X0, changes. The following two equations show how we the STD-RIOC can generate the RD* and WR* signals for the STD bus using the four bit counter for timing information:

RD*=((STB*)'(WT*)(t9+t10+t11+t12+t13+t14+t15+t0))'
WR*=((STB*)'(WT*)'(t9+t10+t11+t12+t13+t14+t15+t0))'

Although this is certainly workable from a logic standpoint, it is not the simplest implementation in terms of the hardware resources available to use. However, at this point I will stop for this month. Next month I will discuss how we actually implement the RD* and WR* signals using flip-flops (you have probably been doing mental flip-flops the whole time you have been reading this). Also, we have left the RIOC master hanging there waiting for XACK*, but it will have to wait till next month too.
+++

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

Star-K Software Systems Corp.
P. O. Box 209
Mt. Kisco, N. Y. 10549
(914) 241-0287

Don Williams
'68 Micro Journal
5900 Cassandra Smith Road
Hixson, TN 37343

Dear Don:

Just thought I'd let you know of two 68xx(x) Bulletin Board Systems.

To provide a way to keep in touch with my many customers and friends in the world, I recently set up a BBS at (914) 241-3307. It is open to all 68xx(x) users, but to avoid clogging up the system, we try to downplay coverage of those systems which are already covered by other bulletin boards or Compuserve forums (such as CoCo's, Macs, Ataris, Amigas, OS-9, or other mass-market computers.)

There is also a Motorola Freeware BBS at (512) 440-3733. It covers a broad spectrum of Motorola processors, including the 68HC11 and 6805 series, and has a number of downloadable files.

Many thanks and best regards.

Sincerely,

Peter A. Stark
President

GESPAC inc.
50 West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Fax. (602) 962-5750

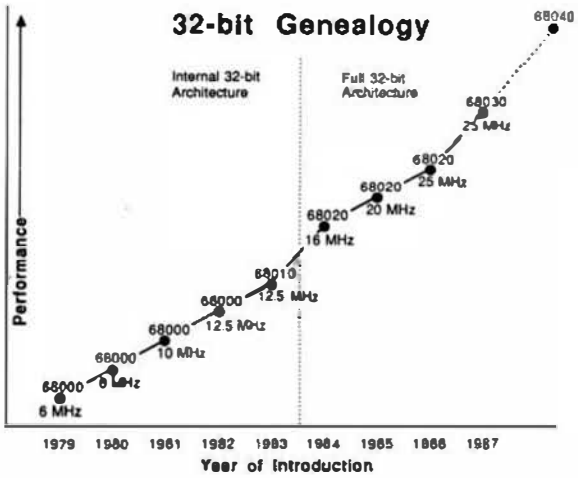
Reader Contact: Mark Stephens
Editorial Contact: Cosma Pabouctsidis

8 CHANNEL RESOLVER INTERFACE BOARD FOR THE G-64 BUS

MESA, AZ, DECEMBER 16, 1987--GESPAC has released a resolver interface board for the G-64 bus. The GESREV-1 conditions and multiplexes eight resolvers and converts their output to a 12 bit number in 400 microseconds per channel. A resolver is a precision inductive angular displacement transducer widely used in robotic and machine tool applications.

The GESREV-1 is equipped with a synchronization mechanism that allows two or more boards to work together in a multi-resolver system. This function allows the user to perform high accuracy readings using a coarse/fine encoding scheme. The excitation for the resolver is provided by an on-board power sinewave oscillator at 2.5 kHz. The board is built on a compact and rugged single height Eurocard of 100 x 160 mm compatible with the standard G-64 bus. The G-64 bus is an easy-to-interface, nonmultiplexed 16 bit bus aimed at industrial microcomputer applications.

The GESREV-1 is available today and is priced at $895.



68000 : First shipped in 1979, the 68000 offers a 32-bit internal architecture on a 16-bit data bus. The 68000 is popular in personal computers, laser printers, robotics, communications and other embedded controllers.

68010 : Added full virtual memory and virtual machine capability to the 68000 architecture, allowing the use of a mass market microprocessor in workstations and parallel computers, pushing computing power to the desktop.

68020 : The first full 32-bit microprocessor. The 68020 is the standard for high-performance workstations and graphics applications. The installed base of over 1 million 68020's in personal computers, workstations, telephone switching systems, factory automation devices, and embedded controllers is supported by a 32-bit software base exceeding $2 billion.

68030 : A second-generation 32-bit microprocessor, the 68030 increases the range of applications suited for 32-bit microprocessors, from inexpensive personal workstations to super computers.

68040 : The third-generation 32-bit microprocessor is now being designed, and will continue the 68000 family's leading performance while maintaining software compatibility

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

For further information contact:

EDITORIAL CONTACT:
Alan Kelly
Cunningham Communication, Inc.
2350 Mission College Boulevard
Suite 900
Santa Clara, CA 95054
(408) 982-0400

READER CONTACT:
Dean Mosley
(512) 440-2839

INQUIRY RESPONSE:
Technical Information Center
P.O. Box 52073
Phoenix, AZ 85072

MOTOROLA UNVEILS 68030

Second-Generation 32-Bit Chip Doubles Performance of 68020
030 to Fuel High-Performance, Low-Cost Applications

NEW YORK, Oct. 29, 1987—At a news conference today, Motorola unveiled its newest 32-bit microprocessor, the 68030 (030), with top executives from Apple Computer, NCR, Northern Telecom, Sun Microsystems and Unisys on hand to endorse Motorola's evolving 68000 line of microprocessors. Motorola has completed six months of evaluation sampling with key customers, and is now accepting orders for the 030 with a top speed of 20 MHz.

The 030, the newest member of the 68000 family, provides twice the performance of the most powerful and widely used 32-bit chip, Motorola's 68020 (020), as well as processors such as Intel Corp.'s 80386. The 030 is the first microprocessor to have on-chip data and instruction caches, parallel (Harvard-style) architecture and dual modes of address. According to Motorola, these advanced features will enable system manufacturers to offer complete 32-bit computer systems with prices as low as $2,000.

"The 030 marks the complete commercialization of 32-bit computing," said Dr. Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "32-bit processing—once an exclusive high-end technology—was made available to many through 020-based systems from Apple, NCR, Sun and Apollo. The 030's features will make 32-bit processing truly a mass-market standard."

Motorola and Logic Automation (Beaverton, Ore.) also announced today development tools for designers creating systems around the new chip (see details in companion releases). Products based on the 030 are expected to be announced early in 1988.

030—Twice the Microprocessor

The 030 (nicknamed "oh thirty") retains all essential features of the 020, and includes a number of enhancements that increase the processor's "parallelism"—the number of functions it can perform simultaneously—and its "bandwidth"—the rate at which it can feed information to its central execution unit.

These innovations not only increase processor performance, they help to reduce overall system cost. Because the 030's high performance does not require extra components such as graphics coprocessors, memory management hardware and expensive static random access memory (SRAM), overall system cost can be reduced while offering the highest performance available. The price of some 030-based systems will be as low as $2,000—a fraction of the cost of current 32-bit systems.

The 030 is the first general-purpose microprocessor with on-chip "cache" memory for computer instructions and data (the 020 was the first with an instruction cache). Cache is very high-speed temporary storage of the information most recently used by the processor. The processor stores this information on-chip in the likely event that it will be used by the execution unit (independent benchmarks have proven that on-chip cache increase processor efficiency by 20 to 40 percent). By storing this essential information on the chip itself the 030 avoids the delays associated with external memory devices.

The 030 is also the first chip with an internal parallel architecture called Harvard-style architecture. The 030 has two independent address buses and two independent 32-bit data buses (See Diagram 1), allowing the execution unit to access and use multiple data sources simultaneously. Traditionally restricted to mainframe computers, supercomputers and reduced instruction set computers (RISC), this architecture provides multiple, parallel data paths on the chip, speeding information flow. (For more information, see the backgrounder entitled: "Motorola's 68030 Second-Generation 32-bit Microprocessor.")

The 030 will boost the computing horsepower of existing systems, such as high-performance scientific and engineering workstations, telecommunication switches and multiprocessor systems (such as supercomputers). It will also serve as the basis for new low-cost 32-bit platforms such as personal workstations for business, engineering and education.

Full Compatibility

The 030 is a fully compatible member of Motorola's M68000 family, which includes the 68000 and the 68020 microprocessors. The 020 is the world's best-selling 32-bit microprocessor. Together with other members of the 68000 family, it has generated the largest 32-bit software base ($2 billion) and most established hardware base ($88 billion). The 030's complete compatibility with preceding 68000-family processors will allow customers to easily take advantage of applications software designed for earlier systems and provide simple hardware upgrades.

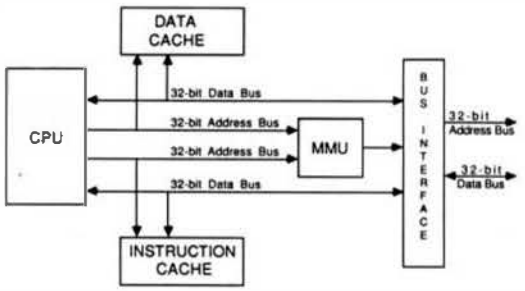
Hundreds of manufacturers currently offer 020-based products. It is the leading 32-bit microprocessor in multiuser business systems, technical workstations, telecommunications, multiprocessing systems, robotics and industrial control. Products based on the 68000-family microprocessors include business computers from Apple, NCR and Unisys; laser printers from Apple, Canon and Ricoh; and engineering workstations from Sun Microsystems and Apollo Computer.

"The ability to span widely varying markets is a testimony to the flexibility and power of the 68000 family of microprocessors," said Goldman. "Our chips are found in everything from airline reservation systems to business computers to supercomputers, and we expect the 030 to expand our presence even further."

Motorola has completed six months of evaluation sampling with key customers is now accepting orders for the 030 at a top speed of 20 MHz. The 030 is currently available at 16 and 20 MHz speeds, with single units priced at $400 and $550 respectively.

Motorola's $2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America with a balanced product portfolio of over 50,000 devices.

68030 INTERNAL BLOCK DIAGRAM



CPU - The General Processing Unit executes the instructions in a program. The CPU gets its instructions and the data required by those instructions from memory. In most computers, CPU performance is limited by the speed at which it is able to obtain these instructions from memory. Two solutions are to use faster memories (which increases system cost) or add on-chip resources around the CPU that speed information flow. The 68030 maintains full compatibility with 32-bit software written for previous 68000 family processors.

32-bit buses - Most 32-bit CPU's have a single 32-bit internal address bus (path through which information moves) and a single 32-bit internal data bus to handle information. The 68030 is the first microprocessor to have two sets of these buses internally on a single chip. The dual buses on the 68030, commonly referred to as a Harvard-style architecture, first appeared in mainframe computers to boost performance.

Instruction Cache - A relatively small, high speed, on-chip memory which holds the most recently used instructions for future re-use by the CPU. This cache, found on the 68020 and 68030, increases performance up to 40% by allowing immediate access to recently used instructions.

Data Cache - A relatively small, high speed, on-chip memory which holds the most recently used data for future re-use by the CPU. This cache, new on the 68030, increases performance by allowing immediate access to recently used data. The 68030 is the first microprocessor to feature both an on-chip instruction and on-chip data cache.

MMU - The Memory Management Unit provides protection by allowing programmers to use system resources without regard for hardware imposed restrictions (e.g. the number of megabytes of actual memory). The MMU also allows multiple programs and/or operating systems to be used simultaneously.

Bus Interface - The bus interface controls all communication between the CPU and external devices. The 68030's bus interface has several unique features allowing optimal access to external SRAMs, DRAMs, and other peripheral devices.

All of these elements operate in parallel, thereby allowing more work to be done than in conventional implementations of microprocessors. For example, the 68030's CPU can simultaneously access the on-chip instruction cache, on-chip data cache, and external memory. The 68030 is the first microprocessor with this capability.

MOTOROLA DISCLOSES DEVELOPMENT OF 68040

Announcement Continues Evolution of 68000 Family, Coincides with Debut of 68030

NEW YORK, Oct. 29, 1987—Motorola today announced the development of a third-generation 32-bit microprocessor, signaling customers that the 68000 microprocessor line will continue to increase in performance. The new chip, called the 68040 (040), will be fully compatible with preceding generations of the 68000 family.

The disclosure of the 040 (nicknamed "oh forty") development project came at the debut of the company's newest 32-bit microprocessor, the 68030 (030) in New York. Motorola announced that it is now taking orders for delivery of the 030. Motorola has not released specifications, pricing or a production schedule for the 040.

"The 68000 family has a flawless eight-year track record, and our customers depend on the compatibility our chips provide to ensure the continuing evolution of their product lines," said Dr. Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "The 68040 will have the characteristics that our customers demand: 68000 family compatibility, leading performance, 32-bit architecture and a secure supply source. As we continue to increase the performance of the 68000 line, our customers know that their next-generation system will not be their last."

M68000 Family Description

The M68000 line currently has four microprocessor members—the 68000, 68010, 68020 and 68030. In total, 10 million chips from the family have been sold. All generations of the 68000 are completely compatible with each other; software written for one chip runs with no modification on the others, and hardware upgrades are very simple. (For more information, see the backgrounders entitled: "Motorola's 68020 and the 32-bit Microprocessor Market" and "Motorola's 68030 Second Generation 32-bit Microprocessor.")

The 68000 chip, introduced in 1979, was the flagship product of the family. Although it was technically classified as a 16-bit microprocessor, its 32-bit internal architecture provided upward compatibility with its true 32-bit successors. The popular chip is found in computers, laser printers, telephone systems and workstations.

The 68010, introduced in 1983, is a modified version of the 68000 chip. It was the first microprocessor to offer "virtual machine" capability, a feature that made it the processor of choice in fault-tolerant computers designed to run without interruption. These include banking systems, transaction terminals, flight reservations systems and communications networks.

The 68020 (020)—the first true 32-bit microprocessor—started a revolution in computing. Introduced in 1984, its unparalleled performance, built-in graphics capability and 32-bit architecture made it the most widely used 32-bit microprocessor in the world. More than 100 companies currently rely on the 020 to power business, scientific and industrial systems. The chip is the foundation of the engineering workstation market and ushered in the graphics revolution; 88 percent of all workstation vendors use the 020. Overall, the 020 represents 72 percent of 32-bit microprocessors sold, beating out all competitors, including the Intel 80386. Before the introduction of the 030, the 020 ranked as the most powerful general-purpose microprocessor on the market.

The 68030 (030), shipping in 1987, is the newest member of the 68000 family. The success of previous generations provide users of the 030 with numerous advantages. It has the largest 32-bit software base ($2 billion) and most established hardware base ($88 billion). The 030 is a significant advance in semiconductor technology. It is the first general-purpose microprocessor to have on-chip data and instruction caches, parallel (Harvard-style) architecture and dual modes of address. Motorola feels that these advanced features will enable system manufacturers to offer complete 32-bit computer systems at prices as low as $2,000.

Motorola's $2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America, with a balanced product portfolio of over 50,000 devices.

MOTOROLA ANNOUNCES 68882 MATH COPROCESSOR

882 Delivers Twice the Performance of 881, Retains 100 Percent Software/Pin Compatibility
General Delivery Begins December 1987

NEW YORK, Oct. 29, 1987—Motorola today announced the 68882 (882), the company's second-generation 32-bit floating point math coprocessor. The 882 offers two to four times the performance of the 68881 (881), the first math coprocessor to strictly conform to the IEEE Standard for Binary Floating Point Arithmetic. The 882 incorporates the same strict conformance to the IEEE Standard, plus software compatibility, pin compatibility and broad-based functionality.

The 882 was introduced in tandem with Motorola's 68030 (030) microprocessor today in New York. The 882 coprocessor is a high-performance single chip used for intensive mathematical calculations. Although it can be used with any microprocessor on an 8-, 16- or 32-bit data bus, its architecture is optimized for the high-performance features of the 68020 and 68030. Motorola is accepting orders for the 882 at speeds of 16 and 20 MHz, priced at $245 and $375 respectively.

Floating point math coprocessors are used to speed mathematical calculations in applications ranging from spreadsheet tabulations to mechanical drawing. Because a coprocessor is a hardware solution, it can perform mathematical functions 100 times as fast as software solutions.

The 882 includes a number of features that increase its "parallelism"—that is, the number of things it can do at once. The 882 executes instructions, moves information internally and externally and converts data simultaneously to increase performance.

The 882 has many similarities to its predecessor, the 881. From the standpoint of applications software, the two are compatible, right down to the results generated by calculations. The instruction set supported by both devices is also identical. Finally, the 881 and 882 are pin compatible. Systems utilizing the 881 will gain a 50 percent performance increase by simply unplugging the 881 and replacing it with an 882. With optimized software, the 882 can provide up to a four-fold performance increase.

"Companies that use other microprocessor architectures have often been forced to scrap their computer and software lines each time their source introduces a new chip," said Dr. Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group. "But like the 030 which provides our 020 customers with a smooth upgrade path, the 882 enables 881 users to speed up their systems without losing their initial investment in system design or software."

882 Technical Specifications

- Eight general-purpose, dual-ported floating point data registers, each register supporting a full 80-bits. Dual porting allows for concurrent accessing, execution and conversion of data.
- A conversion control unit that speeds data format transformations from 32 bits to 80 bits and vice versa.
- A 67-bit arithmetic unit, allowing very fast calculations.
- A 67-bit barrel shifter for high-speed shifting operations.
- 46 instructions, including 35 arithmetic functions such as add, subtract, multiply and divide.
- Operation with any host processor on an 8-, 16-, or 32-bit data bus.

Motorola's $2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest semiconductor supplier in North America, with a balanced product portfolio of over 50,000 devices.

NEW BOOK EXPLAINS ASSEMBLY LANGUAGE PROGRAMMING FOR 68000 SYSTEMS

The 68000 microchip has sixteen times the storage capacity of the 8086/8088 adopted by IBM for its compact PC•system. Other companies recognized the powerful capabilities of the 68000 and the increase in sales of the Apple Macintosh, Commodore Amiga, Atari ST and other 68000-based systems is a testimonial to its growing popularity.

ASSEMBLY LANGUAGE PROGRAMMING FOR THE 68000 FAMILY (Wiley; $22.95, paper; January 15, 1987) by Thomas Skinner offers users of a 68000 system all the information and techniques they need to develop and write assembly language programs. The book covers the entire 68000 line, including 68008, 68010, 68012, 68020 and the new and powerful 68030.

Skinner, an assembly language expert, assumes no prior assembly language experience, explaining the basic principles of assembly language as originated by the 68000 chip's designer, Motorola. The programmer will be able to create simple programs after the first five chapters, and complex programs by the end of the book.

Specific functions and techniques of assembly language programming are covered in depth, including:

- basic input/output subroutines,
- data types,
- programming instructions,
- conditional and arithmetic instructions,
- looping,
- linking, and
- debugging.

Program shells for Macintosh, Amiga and Atari ST series as well as separate chapters devoted exclusively to 68101, 68020 and 68030 are included to provide users of these systems with more specialized information.

SPECIAL OFFER

(One Only!)

MUSTANG-020 12 Mhz modified to run
12 or 16 Mhz
Exactly as shown in Mustang-020 ad page 4

This is Don Williams' "personal system", reason for selling - upgraded to 25 Mhz.

Regular price (16 MHZ with 68881) $5,324.80
Complete with OS-9 & C
SAVE first come ONLY $3,895.80
68 Micro Journal - 615 842-4600
Ask for Tom, Larry or Don. This won't last long!

This system has hand-picked components.

ATARI CALL

As most of you know, we are very sensitive to your wishes, as concerns the content of these pages. One of the things that many of you have repeatedly written or called about is coverage for the Atari™ series of 68000 computers.

Actually we haven't been too keen on these systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari machine. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari computers. Especially since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you). Many of you are doing all kinds of interseting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari*. As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. This is the way it has been for over 10 years. We need articles, technical reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it? Drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

Classifieds — As Submitted - No Guarantees

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20 MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.

S+ System with Cabinet, 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $4800.

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900
Qume Sprint 5 - $800.

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.
3 - Dual 8" drive enclosure with power supply. New in box. $125 each.
5 - Siemens 8" Disk Drive, $100 each.
Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.
TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.
SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09CPU Card - $900 complete.
Tom (615) 842-4600 M-F 9AM to 5PM EST

SK★DOS™

The Generic DOS™ for 68000 applications in

- Industrial Control
- Business Use
- Educational Computing
- Scientific Computing
- Number Crunching
- Dedicated Systems
- Turnkey Systems
- Data Collection
- Single-board Computers
- Bus oriented Computers
- Graphics Workstations
- One-of-a kind Systems
- Advanced Hobbyist Use

SK•DOS is a single-user disk operating system for computers using Motorola 32 bit CPUs such as the 68008, 68000, 68010, and 68020. It provides the power of a full DOS, yet is simple and easy to use, and will run on systems from 32K to 16 megabytes. Because **SK•DOS** is easily implemented on a new system, we call it "The Generic DOS" which allows programs written for one system to be run on many others.

SK•DOS comes with over 40 commands and system programs, including a 6809 emulator which allows 68K **SK•DOS** to run application programs and languages developed for 6809 SK•DOS and other systems. Assemblers, editors, and higher level language support are available from third party software vendors and through public domain software.

SK•DOS is available for single-copy or dealer sales, as well as OEM licensing. Single copies cost $125 (inquire as to available systems). Extremely attractive OEM licensing terms are also available. An optional Configuration Kit contains a detailed Configuration Manual and two disks of source code for system adaptation, including source code for a system monitor/debug ROM and other programs useful for adapting SK•DOS to new systems.

SK•DOS
is available from

STAR-K
SOFTWARE SYSTEMS CORPORATION
P.O. BOX 209 · MT. KISCO, NY 10549 · 914/241-0287
TELEX 510601-6774

INDUSTRIAL PASCAL FOR 68000 AND 6809

PCSK is a package that generates code for a 68000 series processor running on a 68000 development system. It includes the compiler, assembler, linker, host debugger, target debugger, and screen editor, all integrated together and controlled by a menu driven shell program. Source code is included for the runtime library and many of the utilities. Host operating systems supported are OS-9/68000 (Microware), PDOS (Eyring Research), and VERSAdos (Motorola).

PXK9 is a package that generates code for a 6809 processor running on a 68000 development system. Includes all of the features of the PCSK package above, except for the host debugger. Host operating system is OS-9/68000.

I WANT IT, WHERE DO I GET IT?

For more information on either of these two products please contact Certified Software, South East Media, or one of our European Licensees.

OEM LICENSEES

Gespac sa, 3, chemin des Aulx, CH-1228 Geneva / Plan-les-Ouates, Switz. TEL (022) 713400, TLX 429989

PEP Elektronik Systeme GmbH, Am Klosterwald 4, D-8950 Kaufbeuren, West Germany. TEL (08341) 8974, TLX 541233.

Eltec Elektronik GmbH, Galileo-Galilei-Strasse, 6500 Mainz 42, Postfach 65, West Germany. TEL (06131) 50031, TLX 4187273.

DISTRIBUTORS

R.C.S. Microsystems Ltd, 141 Uxbridge Road, Hampton Hill, Middlesex, England. TEL 01-9792204, TLX 8951470.

Dr. Rudolf Keil GmbH, Porphystrasse 15, D-6905 Schriesheim, West Germany. TEL 062 03/6741, TLX 465025.

Elsoft AG, Zelgweg 12, CH-5405 Baden-Daettwil, Switzerland. TEL 056-833377, TLX 828275.

Byte Studio Borken, Butenwall 14, D-4280 Borken, West Germany. TEL 02861-2147, TLX 813343.

CERTIFIED SOFTWARE CORPORATION
616 CAMINO CABALLO, NIPOMO, CA 93444
TEL: (805) 929-1395 TELEX: 467013
FAX: (805) 929-1395 (MID-8AM)

SOFTWARE FOR 680x AND MSDOS

SUPER SLEUTH DISASSEMBLERS
EACH 6809-FLEX $101-OS9 $100-UNIFLEX
OBJECT-ONLY versions: EACH $50-FLEX,OS9,COCO
interactively generate source on disk with labels, include xref, binary editing
specify 6800,1,2,3,5,8,9/6502 version or Z80/8080.5 version
OS9 version also processes FLEX format object file under OS9
COCO DOS available in 6800,1,2,3,5,8,9/6502 version (not Z80/8080.5) only
68010 disassembler $100-FLEX,OS9,UNIFLEX,MSDOS,UNIX,SKDOS

CROSS-ASSEMBLERS WITH MACRO CAPABILITIES
EACH $50-FLEX,OS9,UNIFLEX,MSDOS,UNIX,SKDOS 3/$100 ALL/$200
specify: 180x,6502,6801/11,6804,6805,6809,Z8,Z80,8048,8051,8085,68010,32000
modular cross-assemblers in C, with load/unload utilities
sources for additional $50 each, $100 for 3, $300 for all

DEBUGGING SIMULATORS FOR POPULAR 8-BIT MICROPROCESSORS
EACH $75-FLEX $100-OS9 $80-UNIFLEX
OBJECT-ONLY versions: EACH $50-COCO FLEX,COCO OS9
interactive, cycle accurate processors, include disassembly formatting, binary editing
specify for 6800/1, (1/4)6805, 6502, 6809 OS9, Z80 FLEX

ASSEMBLER CODE TRANSLATORS FOR 6502, 6800/1, 6809
6502 to 6809 $75-FLEX $85-OS9 $80-UNIFLEX
6800/1 to 6809 & 6809 to portable ind. $50-FLEX $75-OS9 $60-UNIFLEX

FULL-SCREEN XBASIC PROGRAMS with cursor control
AVAILABLE FOR FLEX, UNIFLEX, AND MSDOS
DISPLAY GENERATOR/DOCUMENTOR $50 w/source, $25 without
MAILING LIST SYSTEM $100 w/source, $50 without
INVENTORY WITH MRP $100 w/source, $50 without
TABULA RASA SPREADSHEET $100 w/source, $50 without

DISK AND XBASIC UTILITY PROGRAM LIBRARY
$50-FLEX $30-UNIFLEX/MSDOS
edit disk sectors, sort directory, maintain master catalog, do disk sorts, renumber some or all of BASIC program, xref BASIC programs, etc.
non-FLEX versions include sort and renumber only

CMODEM TELECOMMUNICATIONS PROGRAM
$100-FLEX,OS9,UNIFLEX,MS-DOS,UNIX,SKDOS
OBJECT-ONLY versions: EACH $50
menu-driven with terminal mode, file transfer, MODEM7, XON-XOFF, etc.
for COCO and non-COCO; drives internal COCO modem port up to 2400 Baud

DISKETTES & SERVICES

5.25" DISKETTES
EACH 10-PACK $7.50-SSSD/SSDD/DSDD
American made, guaranteed 100% quality, with Tyvek jackets, hub rings, and labels

ADDITIONAL SERVICES FOR THE COMPUTING COMMUNITY

CUSTOMIZED PROGRAMMING
we will customize any of the programs described in this advertisement or in our brochure for specialized customer use or to cover new processors; the charge for such customization depends upon the maintainability of the modifications.

CONTRACT PROGRAMMING
we will create new programs or modify existing programs on a contract basis, a service we have provided for over twenty years; the computers on which we have performed contract programming include most popular models of mainframes, including IBM, Burroughs, Univac, Honeywell, most popular models of minicomputers, including DEC, IBM, DG, HP, AT&T, and most popular brands of microcomputers, including 6800/1, 6809, Z80, 6502, 680x0, using most appropriate languages and operating systems, on systems ranging in size from large telecommunications to single board controllers; the charge for contract programming is usually by the hour or by the task.

CONSULTING
we offer a wide range of business and technical consulting services, including seminars, advice, training, and design, on any topic related to computers; the charge for consulting is normally based upon time, travel, and expenses.

Computer Systems Consultants, Inc.
1454 Lotte Lane, Conyers, GA 30207
Telephone 404-483-4570 or 1717

We take orders at any time, but plan long discussions after 6, if possible.

Contact us about catalog, dealer, discounts, and services. Most programs in source: give computer, OS, disk size. 25% off multiple purchases of same program on one order. VISA and MASTER CARD accepted; US funds only, please. Add GA sales tax (if in GA) and 5% shipping.

(UNIFLEX tm Technical Systems Consultants; OS/9 Microware; COCO Tandy MSDOS Microsoft, SKDOS Stark Software)

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles
OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microproceasor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard ☐ VISA ☐

Card #_____ Exp. Date_____

For 1 Year ____ 2 Years ____ 3 Years ____

Enclosed: $_____

Name _____

Street _____

City _____ State ____ Zip _____

Country _____

My Computer Is: _____

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

***Foreign Surface: Add $12.00 per Year to USA Price.**

***Foreign Airmail: Add $48.00 per Year to USA Price.**

***Canada & Mexico: Add $9.50 per Year to USA Price.**

***U.S. Currency Cash or Check Drawn on a USA Bank !**

68 Micro Journal
5900 Cassandra Smith Rd.
POB 849
Hixson, TN 37343

Telephone 615 842-4600
Telex 510 600-6630

Reader Service Disks

Disk- 1	Filesort, Minicat, Minicopy,Minifms. **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2	Diskedit w/ inst.& fixes, Prime, *Prnnod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3	Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Harkness.
Disk- 8	Creat, Lanpher (May 82).
Disk- 9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Tennnal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong),CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist), Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven. Language Recognition Utility, Anderson March1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**StarTrek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.

NOTE:
This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

GMX MICRO-20 PRICE LIST

MICRO 20 (12.5 MHz) W/1 SAB	$2565.00
MICRO 20 (16.67 MHz) W/1 SAB	$2895.00
MICRO 20 (20 MHz) W/1 SAB	$3295.00

OPTIONAL PARTS AND ACCESSORIES

68881 12.5MHz Floating Point Coprocessor	$ 195.00
68881 16.67MHz Floating Point Coprocessor	$ 295.00
68881 20MHz Floating Point Coprocessor	$ 495.00
MOTOROLA 68020 USERS MANUAL	$ 18.00
MOTOROLA 68881 USERS MANUAL	$ 18.00

SBC ACCESSORY PACKAGE (M20-AP) **$1690.00**

The package includes a PC-style cabinet with a custom backpanel, a 25 Megabyte (unformatted) hard disk and controller, a floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches, and all necessary internal cabling. (For use with SAB—90 serial connectors only.)

2nd 5"80 FLOPPY & CABLES FOR M20-AP, ADD	$ 250.00
SECOND 25MB HARD DISK & CABLES, ADD	$ 780.00
TO SUBSTITUTE 50MB HO FOR 25MB HO, ADD	$ 290.00
TO SUBSTITUTE 80MB HO FOR 25MB HO, ADD	$1500.00
TO SUBSTITUTE 155MB FOR 25MB HO, ADD	$2100.00
60MB TEAC STREAMER WITH ONE TAPE	$ 870.00
PKG. OF 5 TEAC TAPES	$ 112.50
CUSTOM BACK PANEL PLATE (BPP-PC)	$ 44.00

I/O EXPANSION BOARDS

16 PORT SERIAL BOARD ONLY (SBC-16S) ..... **$ 335.00**

The SBC-16S extends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

RS232 ADAPTER (SAB-25, SAB-90 or SAB-8M) **$165.00**

The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

60 LINE PARALLEL I/O BOARD (SBC-60P) **$398.00**

The GMX SBC-60P uses three 68230 Parallel Interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

PROTOTYPING BOARD (SBC-WW) **$75.00**

The SBC-WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual Inline Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

I/O BUS ADAPTER (SBC-BA) **$195.00**

The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

ARCNET LAN board w/o Software (SBC-AN) **$475.00**

The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNET modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coaxial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN **$120.00**

GMX MICRO-20 SOFTWARE

020 BUG UPDATE — PROMS & MANUAL **$150.00**

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD $150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS. ALL SHIPPED STANDARD ON 5¼" DISKS, 3½" OPTIONAL IF SPECIFIED.

OS9

OS9/68020 PROFESSIONAL PAK **$850.00**

Includes O.S., "C", uMACS EDITOR, ASSEMBLER, DEBUGGER, development utilities, 68881 support.

OS9/68020 PERSONAL PAK ..... **$ 400.00**

Personal OS-9 systems require a GMX Micro-20 development system running Professional OS-9/68020 for initial configuration.

Other Software for OS-9/68020

BASIC (included in PERSONAL PAK)	$ 200.00
C COMPILER (included in PROFESSIONAL PAK)	$ 750.00
PASCAL COMPILER	$ 500.00

UNIFLEX

UniFLEX	$ 450.00
UniFLEX WITH REAL-TIME ENHANCEMENTS	$ 800.00

Other Software for UniFLEX

UniFLEX BASIC W/PRECOMPILER	$ 300.00
UniFLEX C COMPILER	$ 350.00
UniFLEX COBOL COMPILER	$ 750.00
UniFLEX SCREEN EDITOR	$ 150.00
UniFLEX TEXT PROCESSOR	$ 200.00
UniFLEX SORT/MERGE PACKAGE	$ 200.00
UniFLEX VSAM MODULE	$ 100.00
UniFLEX UTILITIES PACKAGE I	$ 200.00
UniFLEX PARTIAL SOURCE LICENSE	$1000.00

GMX EXCLUSIVE VERSIONS, CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.

ABSOFT FORTRAN (UniFLEX)	$1500.00
SCULPTOR (specify UniFLEX or OS9)	$ 995.00
FORTH (OS9)	$ 595.00
DYNACALC (specify UniFLEX or OS9)	$ 300.00

GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

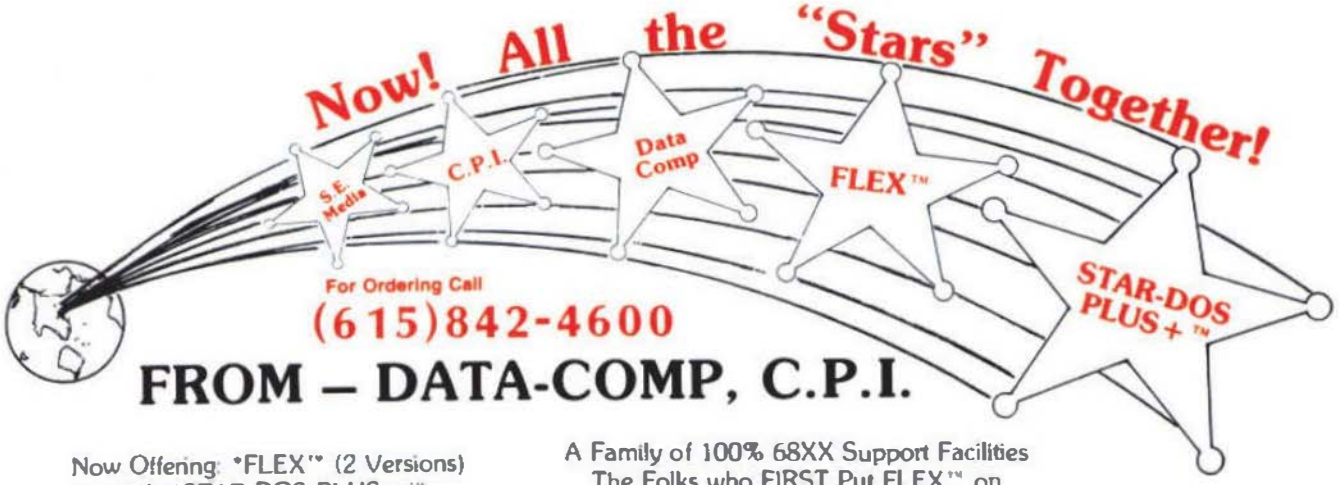
ALL PRICES ARE F.O.B. CHICAGO IN U.S. FUNDS

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS

GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.

GMX™ 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K

FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 systems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SKTDOS™. SKTDOS is a single user, single tasking system that takes up where "FLEX™" left off. SKTDOS is actually a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed *whiz on disk I/O.* Fact is: the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks!

SALE!!

System includes OS-9 68K or SKTDOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K No Wait States	256K Chips
PORTS	4 - RS232 2 - 8 bit Parallel	MC68681 DUART MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details

Now even faster! with 12 Mhz CPU

C Compile times: OS-9 68K	Hard Disk
MUSTANG-08 8 Mhz CPU	0 min - 32 sec
Other popular 68008 system	1 min - 05 sec
MUSTANG-020	0 min - 21 sec

♠ 25 Megabyte Hard Disk System

$2,398.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable ROM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC — MUSTANG-08 is a trademark of CPI

MUSTANG-08

LOOK

					Seconds 32 bit Integer	Register Long
Other	68008	8 Mhz	OS-9	68K	18.0	9.0
MUSTANG-08	10 Mhz		OS-9	68K	9.8	6.3

```
Main()
{
                    /* int i; */
C Benchmark Loop    register long i;
                    for (i=0; i < 999999; ++i);
}
```

Data-Comp Division

CPI — A Decade of Quality Service™ — Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

* Those with SWTPC hi-density FLEX 5" - Call for special info.